NASA Conference Publication 3251

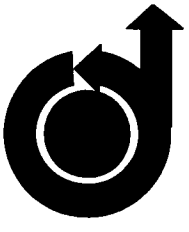
Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS '94)

Volume II

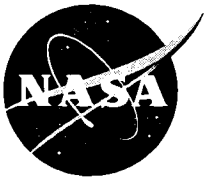
(March 23-24)

Proceedings of a conference sponsored by
The American Institute of Aeronautics & Astronautics
and the National Aeronautics and Space Administration,
Lyndon B. Johnson Space Center,
Houston, Texas

March 21-24, 1994



The American Institute of
Aeronautics and Astronautics



National Aeronautics and
Space Administration

NASA Conference Publication 3251

Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS '94)

Volume II

(March 23-24)

Jon D. Erickson, Editor
NASA Lyndon B. Johnson Space Center
Houston, Texas

Proceedings of a conference sponsored by
The American Institute of Aeronautics & Astronautics
and the National Aeronautics and Space Administration,
Lyndon B. Johnson Space Center, Houston, Texas
March 21-24, 1994

In cooperation with
American Association of Artificial Intelligence,
IEEE Robotics and Automation Society,
National Service Robot Association, Robotic Industries Association,
Society of Optical Instrumentation Engineers,
AIAA Space Automation and Robotics Technical Committee,
Clear Lake Council of Technical Societies, AIAA Houston Section,
ISA Clear Lake Section, and IEEE Galveston Bay Section

National Aeronautics and
Space Administration

Preface

The formation of the AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS '94) was originally proposed because of the strong belief that America's problems of global economic competitiveness and job creation and preservation can partly be solved by the use of intelligent robotics, which are also required for human space exploration missions. It was also recognized that in the applications-driven approach there are a far greater set of common problems and solution approaches in field, factory, service, and space applications to be leveraged for time and cost savings than the obvious differences in implementation details would lead one to believe. This insight, coupled with a sense of national urgency, made a continuing series of conferences to share the details of the common problems and solutions across these different fields of application not only a natural step, but a necessary one. Further, it was recognized that a strong focusing effort is needed to move from recent factory-based robot technology into robotic systems with sufficient intelligence, reliability, safety, multi-task flexibility, and human/machine interoperability to meet the rigorous demands of each of these fields of application. The scope of this effort is beyond the capability of the private sector alone, government alone, or academia alone. Cooperation by all interested parties is essential to achieve the needed investments and maximize the benefits from innovation.

The first AIAA/NASA conference on intelligent robotics is a clear success, judging from the quality and number of papers for presentation and manuscripts collected in these proceedings. Also, having the proceedings available at the conference is important to communication effectiveness and efficiency; the authors are to be congratulated for meeting the deadline. Having Dr. Joseph Engelberger, Chief Executive Officer of Transitions Research Corporation, present the keynote address emphasizing the applications-driven approach to technology development sets the correct tone and background for getting on with the job of strategic investment in and development of intelligent robotics through cooperative national efforts.

The papers in these proceedings are evidence that users in each field, manufacturers and integrators, and technology developers are rapidly increasing their understanding of the "whats" and "hows" of integrating robotic systems on Earth and in space to accomplish economically important tasks requiring mobility and manipulation. The 21 sessions of technical papers in seven tracks plus two plenary sessions cover just the tip of this major progress, but reveal its presence nonetheless.

The contents pages of these proceedings do not necessarily reflect the final program nor the arrangement of presentations in sessions. The conference brochure provides the information.

Appreciation goes to the Steering Committee members, Program Committee members, Track chairs, and Session chairs who are all so essential to making this a successful conference through the voluntary giving of their time and efforts. Special thanks and personal admiration go to Larry Seidman, Zafar Taqvi, Hatem Nasr, Mary Stewart, Donna Maloy, and Dottye Hamblin for their efforts to make this conference happen.

Conference Chair
Jon D. Erickson

General Chair

Paul J. Weitz
NASA Johnson Space Center

Conference Chair

Jon D. Erickson
NASA Johnson Space Center

Technical Program Chair

Lawrence P. Seidman
The MITRE Corporation

Administrative Committe Chair

Zafar Taqvi
Hernandez Engineering

Conference Steering Committe Members

Lawrence Seidman, The MITRE Corporation
R. Peter Bonasso, The MITRE Corporation
Jeffrey Burnstein, National Service Robot
William Hamel, Oak Ridge National Laboratories
John Holland, Cybermotion
Michael Kearney, McDonnell Douglas
Michael Leahy, Kelly Air Force Base
Joseph Loibl, Ford Motor Company - AMTAC
Harvey Meieran, PHD Technologies
Hatem Nasr, Honeywell
Joseph Parrish, NASA Headquarters
Arthur Sanderson, Rensselaer Polytechnic Institute
Zafar Taqvi, Hernandez Engineering
Delbert Tesar, University of Texas at Austin
Richard Volz, Texas A&M University

Technical Program Committee Members

R. Peter Bonasso, The MITRE Corporation
John Borgman, University of Texas at Austin
Andy Chang, Ford Motor Company
Francis daCosta, ACG
Mark Gittleman, Oceaneering
Raymond Harrigan, Sandia National Laboratory
William Hamel, Oak Ridge National Laboratories
Butler Hine, NASA Ames Research Center
John Holland, Cybermotion
Steven Holland, General Motors
David Hunter, Canadian Space Agency
James Karlen, Robotics Research
Michael Kearney, McDonnell Douglas
Michael Leahy, Kelly Air Force Base
Paul Mattaboni, Cyberotics
Harvey Meieran, PHD Technologies
Jay Mendelson, Grumman
Hatem Nasr, Honeywell
Robert Palmquist, Sandia National Laboratory
Joseph Parrish, NASA Headquarters
Harold Roman, Public Service Electric & Gas
Arthur Sanderson, Rensselaer Polytechnic Institute
Robert Savely, NASA Johnson Space Center
Delbert Tesar, University of Texas at Austin
Richard Theobald, Lockheed Engineering & Sciences
Richard Volz, Texas A&M University
Carl Weiman, Transitions Research
Charles Wu, Ford Motor Company

Contents

Volume I
(March 21-22)

First Plenary Session: *Jon D. Erickson, Conference Chair*

Intelligent Robotics Can Boost America's Economic Growth
Jon D. Erickson, *Chief Scientist for Automation and Robotics Division, NASA Johnson Space Center, Houston, Texas* 1

Field Track: *Harvey B. Meieran, Chair*

Nuclear Industry Session: Jack J. Judge, Jr., Chair

Teleoperated Systems for Nuclear Reactors Inspection and Maintenance
V. P. Dorokhov, *TECHNIKA, St. Petersburg, Russia*; D. V. Dorokhov, *Baltic State Technical University, St. Petersburg, Russia*; A. P. Eperin, *Leningrad Atomic Power Station, Sosnoviy Bor, Russia* 8

ARK: Autonomous Mobile Robot in an Industrial Environment
S. B. Nickerson, J. R. R. Service, and D. Wilkes, *Ontario Hydro Technologies, Toronto, Canada*; P. Jasiobedzki, A. Jepson, B. Down, D. Terzopoulos, and J. Tsotsos, *University of Toronto, Toronto, Canada*; M. Jenkin and E. Milios, *York University, North York, Canada*; N. Bains and T. Campbell, *Atomic Energy of Canada Ltd., Mississauga, Canada* 12

Biologically-Inspired Hexapod Robot Design and Simulation
Kenneth S. Espenschied and Roger D. Quinn, *Case Western Reserve University, Cleveland, Ohio* 21

Odysseus Autonomous Walking Robot: The Leg/Arm Design
N. G. Bourbakis, M. Maas, A. Tascillo, and C. Vandewinckel, *Binghamton University, Binghamton, New York* 29

Factory Track: *Michael B. Leahy, Jr., Chair*

Agile Manufacturing Session: Joseph M. Loibl, Chair

A Vision Advisor System for Flexible Manufacturing
Jim Eilbert, William Lim , Jay Mendelsohn, Ron Braun and Mike Yearwood, *Grumman Corporation, Bethpage, New York* 37

Application of the Modular Automated Reconfigurable Assembly System Concept to Adaptable Vision Gauging and Parts Feeding
Andre By and Ken Caron, *Tufts University, Medford, Massachusetts*; Michael Rothenberg and Vic Sales, *Productivity Technologies, Inc., Sunnyvale, California* 47

Agile Manufacturing and The Factory of the Future
Joseph M. Loibl and Terry Bossieux, *Ford Motor Co. - AMTAC, Dearborn, Michigan* 56

Contents
(continued)

Confessions of a Robot Lobotomist
R. Marc Gottshall, *Boeing Company, Commercial Airplane Group, Seattle, Washington* 63

Integration of Vision and Robotic Workcell
Terry Bossieux, *Ford Motor Co. - AMTAC, Dearborn, Michigan* 73

Service Track: *John M. Holland, Chair*

Security/Building Monitoring Session: *Celeste DeCorte, Chair*

Meeting the Challenges of Installing a Mobile Robotic System
Celeste DeCorte, *Cyberomotion, Inc., Salem, Virginia* 78

Design of an Autonomous Exterior Security Robot
Scott D. Myers, *Robotic Systems Technology, Westminster, Maryland* 82

Task Automation in a Successful Industrial Telerobot
Sammy L. Jones, *Remotec, Inc., Oak Ridge, Tennessee*; Phillip F. Spelt, *Oak Ridge National Laboratory, Oak Ridge, Tennessee* 88

Controlling Multiple Security Robots in a Warehouse Environment
H. R. Everett, G. A. Gilbreath, T. A. Heath-Pastore, and R. T. Laird, *Naval Command Control & Ocean Surveillance Center, San Diego, California* 93

Space Track: *Joseph C. Parrish, Chair*

On-Orbit Applications I Session: *Joseph C. Parrish, Chair*

Technology Transfer and Evaluation for Space Station Telerobotics
Charles R. Price and Lebarian Stokes, *NASA Johnson Space Center, Houston, Texas*; Myron A. Diftler, *Lockheed Engineering & Sciences Company, Houston, Texas* 103

Space Flight Manipulator Technologies and Requirements as Developed for the NASA Flight Telerobotic Servicer (FTS)
John T. Chladek, *NASA Johnson Space Center, Houston, Texas*; William M. Craver, *Lockheed Engineering & Sciences Company, Houston, Texas* 112

A Space Station Robot Walker and Its Shared Control Software
Yangsheng Xu and Ben Brown, *Carnegie Mellon University, Pittsburgh, Pennsylvania*; Shigeru Aoki and Tetsuji Yoshida, *SHIMIZU Corporation, Tokyo, Japan* 123

Technology for Robotic Surface Inspection in Space
Richard Volpe and J. Balaram, *NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California* 131

A Highly Redundant Robot System for Inspection
Thomas Lee, Tim Ohms, and Samad Hayati, *NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California* 142

Free-Floating Dual-Arm Robots for Space Assembly
Sunil K. Agrawal and M. Y. Chen, *Ohio University, Athens, Ohio* 149

Contents
(continued)

Robotic Sensing, Vision, and Perception Track: *Hatem Nasr, Chair*

Vision & Sensing Technologies for Autonomous Robots Session: *Bir Bhanu, Chair*

Design and Control of Active Vision Based Mechanisms for Intelligent Robots
Liwei Wu and Michael M. Marefat, *University of Arizona, Tucson, Arizona* 158

Vision Based Object Pose Estimation for Mobile Robots
Annie Wu, Clint R. Bidlack, Terry Weymouth, Arun Katkere, and Roy Feague, *The University of Michigan, Ann Arbor, Michigan* 168

Unsupervised Texture Image Segmentation by Improved Neural Network ART2
Zhiling Wang, G. Sylos Labini, and R. Mugnuolo, *Italian Space Agency, Matera, Italy*; Marco De Sario, *University of Bari, Bari, Italy* 175

Microwave Vision for Robots
Leon Lewandowski and Keith Struckman, *Lockheed Sanders, Inc., Nashua, New Hampshire* 181

An Electromagnetic NonContacting Sensor for Thickness Measurement in a Dispersive Media
Robert L. Chufo, *U.S. Dept. of the Interior, Bureau of Mines, Pittsburgh, Pennsylvania* 190

Perception System and Functions for Autonomous Navigation in a Natural Environment
Raja Chatila, Michel Devy, Simon Lacroix, and Matthieu Herrb, *Centre National de la Recherche Scientifique, Laboratoire d'Automatique et d'Analyse des Systems, Toulouse, France* 195

Planning, Reasoning, and Control Track: *R. Peter Bonasso, Chair*

Situated Control and Low-Level Control Session: *William Lim, Chair*

Fuzzy Logic Based Robotic Controller
Farouk G. Attia, M. Upadhyaya, and S. Ramchandani, *University of Houston, Houston, Texas* 206

Vehicle Following Controller Design for Autonomous Intelligent Vehicles
C. C. Chien, R. Mayr, and M. C. Lai, *University of Southern California, Los Angeles, California* 212

The Real-World Navigator
Illah R. Nourbakhsh, Craig Becker, Marko Balabanovic, and Sarah K. Morse, *Stanford University, Stanford, California* 223

A Streamlined Software Environment for Situated Skills
Sophia T. Yu, Marc G. Slack, and David P. Miller, *The MITRE Corporation, McLean, Virginia* 233

Situational Reaction and Planning
John Yen and Nathan J. Pfluger, *Texas A&M University, College Station, Texas* 240

Contents
(continued)

Autonomous Mobile Robot Teams
Arvin Agah and George A. Bekey, *University of Southern California, Los Angeles, California* 246

Systems Technology and Architectures Track: *Arthur C. Sanderson, Chair*

Robotic Systems Architectures Session: *Ian D. Walker, Chair*

Building Brains for Bodies
Rodney Allen Brooks and Lynn Andrea Stein, *Massachusetts Institute of Technology, Cambridge, Massachusetts* 253

Object-Based Task-Level Control: A Hierarchical Control Architecture for Remote Operation of Space Robots
Homer D. Stevens, Eric S. Miles, Stephen M. Rock, and Robert H. Cannon, *Stanford University, Stanford, California* 264

Task-Level Control for Autonomous Robots
Reid Simmons, *Carnegie Mellon University, Pittsburgh, Pennsylvania* 275

A Survey of NASA and Military Standards on Fault Tolerance and Reliability Applied to Robotics
Joseph R. Cavallaro and Ian D. Walker, *Rice University, Houston, Texas* 282

A Performance Analysis Method for Distributed Real-Time Robotic Systems: A Case Study of Remote Teleoperation
Donald R. Lefebvre and Arthur C. Sanderson, *Rensselaer Polytechnic Institute, Troy, New York* 287

Predictive Sufficiency and the Use of Stored Internal State
David J. Musliner, *The University of Maryland, College Park, Maryland*; Edmund H. Durfee, and Kang G. Shin, *The University of Michigan, Ann Arbor, Michigan* 298

Using Generic Tool Kits to Build Intelligent Systems
David J. Miller, *Sandia National Laboratories, Albuquerque, New Mexico* 306

Field Track: *Harvey B. Meieran, Chair*

Environmental Restoration, Waste Management, and Hazardous Operation Session: *Harvey B. Meieran, Chair*

A Reactive System for Open Terrain Navigation: Performance and Limitations
Dirk Langer, Martial Hebert, and J. K. Rosenblatt, *Carnegie Mellon University, The Robotics Institute, Pittsburgh, Pennsylvania* 315

The Road Plan Model - Information Model for Planning Road Building Activities
Rafaela K. Azinhal, *UNINOVA, Monte da Caparica, Portugal*; Fernando Moura-Pires, *Universidade Nova de Lisboa, Monte da Caparica, Portugal* 322

Design Reuse Experience of Space and Hazardous Operations Robots
P. Graham O'Neil, *Lockheed Engineering & Sciences Company, Houston, Texas* 333

Contents
(continued)

A Multi-Mode Manipulator Display System for Controlling Remote Robotics Systems
Michael J. Massimino, M. F. Meschler, and A. A. Rodriguez, *McDonnell Douglas Aerospace, Houston, Texas* 339

Factory Track: *Michael B. Leahy, Jr., Chair*

Robotic Remanufacturing Session: *Michael B. Leahy, Jr., Chair*

Programmable Automated Welding System
Martin D. Kline, *Lynchburg, Virginia*; Thomas E. Doyle, *Alliance, Ohio* 348

Robotic NDE Inspection of Advanced Solid Rocket Motor Casings
Glenn E. McNeelege and Chris Sarantos, *Babcock & Wilcox, Lynchburg, Virginia* 354

Automation For Nondestructive Inspection of Aircraft
M. W. Siegel, *Carnegie Mellon University, Pittsburgh, Pennsylvania* 367

Graphical Simulation for Aerospace Manufacturing
Brian K. Christensen and Majid Babai, *NASA Marshall Space Flight Center, Huntsville, Alabama*; Christopher Bien, *Deneb Robotics, Inc. Auburn Hills, Michigan* 378

The Automated Aircraft Rework System (AARS) -- A System Integration Approach
Michael J. Benoit, *Mercer University, Warner Robins, Georgia* 381

Automated Inspection of Turbine Blades: Challenges and Opportunities
Manish Mehta, Joseph C. Marron, Robert E. Sampson, and George M. Peace, *ERIM, Ann Arbor, Michigan* 393

Service Track: *John M. Holland, Chair*

Healthcare Session: *W. Stuart Lob, Chair*

TRC Research Products: Components for Service Robots
W. Stuart Lob, *Transitions Research Corporation, Danbury, Connecticut* 399

An update on "Lab Rover": a Hospital Material Transporter
Paul Mattaboni, *Cyberotics, Inc., Waltham, Massachusetts* 405

A Robotic Wheelchair
David P. Miller, *The KISS Institute, Reston, Virginia*; Edward Grant, *Power Concepts Incorporated, Fresno, California* 407

Dexterity Enhancement in Microsurgery Using Telemicro-Robotics
Steve Charles andSteve Charles, M. D., *Micro-Dexterity Systems, Inc., Memphis, Tennessee* 412

An Intelligent Robotic Aid System for Human Services
Kazuhiko Kawamura, Sugato Bagchi, and Todd Park, *Vanderbilt University, Nashville, Tennessee* 413

Contents
(continued)

Space Track: *Joseph C. Parrish, Chair*

On-Orbit Applications II Session: *David B. Lavery, Chair*

An Intelligent Robot for Helping Astronauts
Jon D. Erickson, Keith A. Grimm, and Thomas W. Pendleton, *NASA Johnson Space Center, Houston, Texas* 421

Terrestrial Applications of NASA Space Telerobotics Technologies
David B. Lavery, *NASA Headquarters, Washington, D.C.* 434

On-Orbit Spacecraft Servicing - An Element in the Evolution of Space Robotics Applications
Carl J. Anders and Claude H. Roy, *Spar Aerospace Ltd., Brampton, Ontario, Canada* 441

A Modular Artificial Intelligence Inference Engine System (MAIS) for Support of On Orbit Experiments
Thomas M. Hancock III, *New Technology Incorporated, Huntsville, Alabama* 451

Volume II
(March 23-24)

Robotic Sensing, Vision, and Perception Track: *Hatem Nasr, Chair*

Vision Systems Integration and Architecture Session: *Martial Herbert, Chair*

Research on an Autonomous Vision-Guided Helicopter
Omead Amidi, Yuji Mesaki, and Takeo Kanade, *Carnegie Mellon University, Pittsburgh, Pennsylvania* 456

Real-Time Tracking of Objects for Space Applications Using a Laser Range Scanner
Francois Blais, R. A. Couvillon, and Marc Rioux, *National Research Council/IIT, Ontario, Canada*; S. G. MacLean, *Canadian Space Agency, St-Hubert, Quebec, Canada* 464

Integration for Navigation on the UMass Mobile Perception Lab
Edward M. Riseman, Allen R. Hanson, Bruce Draper, and Benny Rochwerger, *University of Massachusetts at Amherst, Amherst, Massachusetts*; Claude Fennema, *Mount Holyoke College, Hadley, Massachusetts* 473

The 4-D Approach to Visual Control of Autonomous Systems
Ernst D. Dickmanns, *Universitaet der Bundeswehr Muncheon, Neubiberg, Germany* 483

Fusion of Laser and Image Sensory Data for 3-D Modeling of the Free Navigation Space
Mark Maas, N. G. Bourbakis, and A. Moghaddamzadeh, *Binghamton University, Binghamton, New York* 494

Contents
(continued)

Planning, Reasoning, and Control Track: *R. Peter Bonasso, Chair*

Selective Perception and Human Robot Interaction Session: *Avi Kak, Chair*

Tele-Assistance for Semi-Autonomous Robots
Robin R. Murphy, *Colorado School of Mines, Golden, Colorado*; Erika Rogers, *Clark Atlanta University, Atlanta, Georgia* 500

Mobile Robot Exploration and Navigation of Indoor Spaces Using Sonar and Vision
David Kortenkamp, *The MITRE Corp., Houston, Texas*; Marcus Huber, Frank Koss, Jaeho Lee, Annie Wu, William Belding, Clint Bidlack, and Seth Rodgers, *The University of Michigan, Ann Arbor, Michigan* 509

The Ground Vehicle Manager's Associate
Gary Edwards, Bruce L. Bullock, Gary R. Edwards, Robert H. Burnard, and William L. Bewley, *ISX Corporation, Westlake Village, California* 520

Deictic Primitives for General Purpose Navigation
Jill D. Crisman and Michael Clearly, *Northeastern University, Boston, Massachusetts* 527

A Methodology for the Generation of the 2-D Map from Unknown Navigation Environment by Traveling a Short Distance
N. G. Bourbakis and D. Sarkar, *Binghamton University, Binghamton, New York* 538

Systems Technology and Architectures Track: *Arthur C. Sanderson, Chair*

Robotic Systems Technologies Session: *Raymond W. Harrigan, Chair*

Simplifying Applications Software for Vision Guided Robot Implementation
Charlie Duncheon, *Adept Technology, Inc., San Jose, California* 543

An Open Architecture Motion Controller
Lothar Rossol, *Trellis Software & Controls, Inc., Rochester Hills, Michigan* 551

Telerobotics for Depot Modernization
Michael B. Leahy, Jr. and S. B. Petroski, *Kelly Air Force Base, San Antonio Air Logistics Center Kelly AFB, Texas* 557

A Smart Telerobotic System Driven by Monocular Vision
Rui J. P. deFigueiredo, Andrea Maccato, and Peter Wlczek, *University of California at Irvine, Irvine, California*; Bradley Denney and John Sheerer, *McDonnell Douglas Aerospace, Huntington Beach, California* 563

GA-Optimization for Rapid Prototype System Demonstration
Jinwoo Kim and Bernard P. Zeigler, *University of Arizona, Tucson, Arizona* 571

A Robot Control Formalism Based on an Information Quality Concept
Anders Ekman, Anders Torne, and Dan Stromberg, *FOA (National Defense Research Establishment), Linkoping, Sweden* 580

Contents
(continued)

Field Track: *Harvey B. Meieran, Chair*

Military and Other Field Applications Session: *John A. Davis, Chair*

The Network Data Delivery Service: A Real-Time Data Connectivity System
Stan Schneider, *Real-Time Innovations, Inc., Stanford University Aerospace Robotics Laboratory, Sunnyvale, California* 591

Robotics in A Controlled Environment Agriculture
Gaines E. Miles, *Purdue University, West Lafayette, Indiana*; Jon D. Erickson, *NASA Johnson Space Center, Houston, Texas* 598

Robotic Hauling Truck for Surface Mining
Keith Chrystall, Patrick Feighan, and Peter Wojcik, *Alberta Research Council, Calgary, Alberta, Canada*; Julian Coward, *Syncrude Canada Ltd., Edmonton, Alberta, Canada*; Ron Eirich and Clement Laforce, *Defence Research Establishment Suffield, Alberta, Canada* 606

A Nonlinear Strategy for Sensor Based Vehicle Path Control
Robert Mayr, *University of Southern California, Los Angeles, California* 614

The Problem with Multiple Robots
Marcus J. Huber and Patrick G. Kenny, *The University of Michigan, Ann Arbor, Michigan* 620

Factory Track: *Michael B. Leahy, Jr., Chair*

Dual-Use Precommercial Robotic Technology Session: *Francis daCosta, Chair*

A Generic Telerobotics Architecture for C-5 Industrial Processes
Wayne Zimmerman and Paul G. Backes, *NASA Jet Propulsion Laboratory, California Institute of Technology Pasadena, California*; Michael B. Leahy, Jr., *Kelly Air Force Base, San Antonio Air Logistics Center Kelly AFB, Texas* 630

A Practical Method of Reverse Engineering and Automatic Path Programming for Robotic Surface Finishing
William T. Adams, John M. Fitzgerald, T. J. Lawley, and O. R. Mitchell, *University of Texas at Arlington, Fort Worth, Texas* 640

Virtual Environments for Telerobotic Shared Control
Brian K. Christensen, *Deneb Robotics, 3285 Lapeer Road West P. O. Box 214687 Auburn Hills, Michigan 48321* 646

Designing the Next Generation of Robotic Controllers
David G. Goldstein, *North Carolina A&T State University, Computer Science Department, Greensboro, North Carolina 27411* 656

Service Track: *John M. Holland, Chair*

Building Operations Session: *Charles "Buck" Ward, Chair*

An End User's Wishlist
Charles W. Ward, *Commercial Service of Virginia, Inc., Richmond, Virginia* 662

Contents
(continued)

The First Commercial Floor Care Company That Ventured into the Production of Robotics
Allen J. Bancroft, *The Kent Company, Elkhart, Indiana* 669

Space Track: *Joseph C. Parrish, Chair*

Planetary Exploration Applications Session: *Brian H. Wilcox, Chair*

Non-Geometric Hazard Detection for a Mars Microrover
Brian H. Wilcox, *NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California* 675

Low Computation Vision-Based Navigation for a Martian Rover
A. S. Gavin, Rodney A. Brooks and Andrew S. Gavin, *Massachusetts Institute of Technology, Cambridge, Massachusetts* 685

The "MITY" Micro-Rover: Sensing, Control, and Operation
Eric Malafeew and William Kaliardos, *Massachusetts Institute of Technology, Cambridge, Massachusetts* 696

Supervised Space Robots Are Needed in Space Exploration
Jon D. Erickson, *NASA Johnson Space Center, Houston, Texas* 705

A Multitasking Behavioral Control System for the Robotic All Terrain Lunar Exploration Rover (RATLER)
Paul Klarer, *Sandia National Laboratories, Albuquerque, New Mexico* 717

Robotic Sensing, Vision, and Perception Track: *Hatem Nasr, Chair*

Vision Systems Integrations and Architecture II Session: *Martial Herbert, Chair*

SAVA III: A Testbed for Integration and Control of Visual Processes
James L. Crowley, *LIFIA-IMAG, Grenoble, France*; Henrik Christensen, *Aalborg University, Aalborg, Denmark* 724

An Architecture for Real-Time Vision Processing
Chiun-Hong Chien, *Lockheed Engineering & Sciences Company, Houston, Texas* 736

Motion Estimation of Objects in KC-135 Microgravity
Lisa Hewgill, *Lockheed Engineering & Sciences Company, Houston, Texas* 744

Real-Time Tracking of Objects for a KC-135 Microgravity Experiment
Mark L. Littlefield, *Lockheed Engineering & Sciences Company, Houston, Texas* 750

Grasping Objects Autonomously in Simulated KC-135 Zero-G
Robert S. Norsworthy, *Lockheed Engineering & Sciences Company, Houston, Texas* 757

Object Tracking with Stereo Vision
Eric L. Huber, *The MITRE Corporation, Houston, Texas*; Kenneth Baker, *NASA Johnson Space Center, Houston, Texas* 763

Contents
(continued)

Planning, Reasoning, and Control Track: *R. Peter Bonasso, Chair*

Planning Session: *R. Peter Bonasso, Chair*

A Software Architecture for Hard Real-Time Execution of Automatically Synthesized Plans or Control Laws
Marcel Schoppers, *Robotics Research Harvesting, Redwood City, California* 768

All Feasible Plans Using Temporal Reasoning
Debasis Mitra and Rasiah Loganantharaj, *University of Southwestern Louisiana, Lafayette, Louisiana* 776

Integrating Deliberative Planning In a Robot Architecture
Christopher Elsaesser and Marc G. Slack, *The MITRE Corporation, McLean, Virginia* 782

Planning in Subsumption Architectures
Eugene C. Chalfant, *University of Southern California, Los Angeles, California* 788

Real-Time Robot Deliberation by Compilation and Monitoring of Anytime Algorithms
Shlomo Zilberstein, *University of Massachusetts at Amherst, Amherst, Massachusetts* 799

Passive Mapping and Intermittent Exploration for Mobile Robots
Sean P. Engelson, *Yale University, New Haven, Connecticut* 810

Systems Technology and Architectures Track: *Arthur C. Sanderson, Chair*

New Directions in Robotic Systems Session: *Samad Hayati, Chair*

Extensibility in Local Sensor Based Planning for Hyper-Redundant Manipulators (Robot Snakes)
Howie Choset and Joel Burdick, *CalTech, Pasadena, California* 820

Fault Tolerant Kinematic Control of Hyper-Redundant Manipulators
Nazareth S. Bedrossian, *C. S. Draper Laboratory, Houston, Texas* 830

Failure Tolerant Operation of Kinematically Redundant Manipulators
Anthony A. Maciejewski and Christopher L. Lewis, *Purdue University, West Lafayette, Indiana* 837

UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications
Jaeho Lee, Marcus J. Huber, Edmund H. Durfee, and Patrick G. Kenny, *The University of Michigan, Ann Arbor, Michigan* 842

Control of Parallel Manipulators Using Force Feedback
Prabjot Nanua, *University of Houston, Houston, Texas* 850

Robust Inverse Kinematics Using Damped Least Squares with Dynamic Weighting
D. E. Schinstock, T. N. Faddis, and R. B. Greenway, *University of Kansas, Lawrence, Kansas* 861

Contents
(concluded)

ControlShell: A Real-Time Software Framework
Stan Schneider, Vincent Chen, and Gerardo Pardo-Castellote, *Real-Time Innovations, Inc., Stanford University Aerospace Robotics Laboratory, Sunnyvale, California* 870

Second Plenary Session: *George Kozmetsky, Chair*

Commercialization

JSC Proposed Dual-Use Technology Investment Program in Intelligent Robotics
Jon D. Erickson, *NASA Johnson Space Center, Houston, Texas* 878

Wednesday
March 23, 1994

AIAA-94-1240-CP

RESEARCH ON AN AUTONOMOUS VISION-GUIDED HELICOPTER

Omead Amidi
Yuji Mesaki
Takeo Kanade
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania

Abstract

We present an overview of the autonomous helicopter project at Carnegie Mellon's Robotics Institute. The goal of this project is to autonomously fly helicopters using computer vision closely integrated with other on-board sensors. We discuss a concrete example mission designed to demonstrate the viability of vision-based helicopter flight and specify the components necessary to accomplish this mission. Major components include customized vision processing hardware designed for high bandwidth and low latency processing and 6-degree-of-freedom test stand designed for realistic and safe indoor experiments using model helicopters. We describe our progress in accomplishing an indoor mission and show experimental results of estimating helicopter state with computer vision during actual flight experiments.

Introduction

Precise maneuverability of helicopters makes them useful for many critical tasks including rescue and security operations, traffic monitoring, mountain fire fighting, and inspection of power transmission lines. The goal of our project is to build a vision-guided helicopter capable of performing these tasks while flying autonomously. In addition to robust helicopter control methods, the development of such a system requires research on vision algorithms for helicopter positioning and object recognition necessary for navigation and tracking tasks, together with real-time hardware for high speed, robust execution of these tasks.

An autonomous helicopter's performance is critically dependent on accurate and frequent estimates of its position and attitude. We focus on methods to provide these estimates using on-board cameras closely integrated with other sensors such as gyroscopes and accelerometers.

Copyright ©1993 American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

We have demonstrated our first results on autonomous helicopter flight. We have built an indoor calibrated testbed that allows free flight experiments with model helicopters. We have custom designed vision hardware which integrates data from on-board sensors with real-time image processing and can now achieve frame-rate (30 Hz) vision-based state estimation. Integrating this vision hardware into a stable control system will lead to outdoor autonomous helicopter flight for performing useful, practical missions.

Motivation

A helicopter is an indispensable air vehicle for emergency operations, such as rescuing stranded individuals and spraying fire extinguishing chemicals for fighting forest fires. Uses of helicopters in the electric power industry include inspecting towers and transmission lines for corrosion and other defects. All of these applications demand dangerous flight patterns in close proximity to the ground or other objects which can risk pilot safety. An unmanned helicopter that operates autonomously or is piloted remotely will eliminate these risks and increase the helicopter's effectiveness.

Typical missions of autonomous helicopters require flying at low speeds to follow a path or hovering near an object. Positioning equipment such as Inertial Navigation Systems (INS) or Global Positioning Systems (GPS) are well suited for long range, low precision helicopter flight and fall short for very precise, close proximity flight. Maneuvering helicopters close to objects requires accurate positioning in relation to the objects. Visual sensing is a rich source of data for this relative feedback.

It is difficult, however, to recover helicopter position and attitude from vision alone. For instance, distinguishing between rotation and translation in a sequence

of images under perspective projection is extremely difficult. On the other hand, the new generation of lightweight gyroscopes and angular rate sensors in the market provide reliable measurement of angular change in an image sequence. For this reason, we concentrate on low-level, close integration of such sensors with vision.

Related Work

The study of the helicopter control problem is not new. Overcoming the inherent instability of helicopters has been the focus of a large body of research, including detailed mathematical models (eg., [10]) for control and Kalman filtering of multiple sensor data for state estimation (eg.,[3]). The controller design methods range from linear quadratic (LQ) design to H^{∞} design [19] and predictive control [8]. For example, a stable closed loop control system has been formulated [3] by quadratic synthesis techniques for helicopter autolanding.

Recently, incorporation of a pilot model has been attempted based on quadratic optimal Cooperative Control Synthesis [17]. This model is used for control augmentation where the control system cooperates with the pilot to increase aircraft performance. The sophisticated pilot model developed by [7] attempts to describe the human's ability to look ahead, which is crucial to precise low-altitude helicopter control. While it is difficult to identify and verify these models, they provide a valuable basis for an intelligent helicopter controller, especially in designing low-level control loops. In this project, we employ a set of low-level controllers which have been designed by using a simplified helicopter dynamics model.

Actual flight tests of helicopter controllers have also been done. Notable implemented systems include those at NASA Ames Research Center [17], NASA Langley Research Center [3], and military aircraft manufacturers [5]. Fuzzy controllers have been successfully employed for actual helicopter flight experiments. In Japan, Sugeno's group at Tokyo Institute of Technology [14] has demonstrated fuzzy control of helicopters for crop dusting.

The state feedback for the above helicopter control experiments was primarily provided by on-board INS/GPS or ground-based beacon systems instead of on-board computer vision. Recently, we are beginning to see promising results in real-time vision-based processors, visual servoing of robotic manipulators, and accurate vision-based position estimation systems, some of which are applicable to autonomous helicopter control experiments.

The development of low-cost special-purpose image correlation chips and new multi-processor architectures capable of high communication rates has made a great impact on image processing. Examples of vision systems built from this kind of hardware include transputer-based image hardware for two-dimensional object tracking [4] and real-time tracking and depth map generation using correlation chips [9].

The high rate of image processing has made inclusion of visual feedback in servo loops practical. There is significant development in visual control of manipulators carrying small cameras, eye-in-hand configuration. Researchers at Carnegie Mellon's Robotics Institute [12] demonstrated real-time visual tracking of arbitrary 3D objects traveling at unknown 2D velocities using a direct-drive manipulator arm. The Yale spatial robot juggler [13] demonstrated transputer-based stereo vision for locating juggling balls in real time. Real-time tracking and interception of objects using a manipulator [11] has also been demonstrated based on fusion of the visual feedback and acoustic sensing.

Controlling by vision requires position estimation relative to desired objects and extraction of 3D scene structure based on sequence of images. RAPiD and DROID [6], developed by Roke Manor Research Limited, are systems designed for performing such tasks in unknown environments. RAPiD is a model-based tracker capable of extracting the position and orientation of known objects in the scene. DROID is a feature-based system which uses the structure-from-motion principle for extracting scene structure using image sequences. Real-time implementations of these systems have been demonstrated using dedicated hardware.

Integrating efficient model-based and connectionist techniques with powerful hardware architectures has produced an array of autonomous land and air vehicles. Significant advances in autonomous automobiles has demonstrated vision-based control at highway speeds. Most notable are Carnegie Mellon's Navlab [16] project and the work of Dickmanns at University of Bundeswehn, Munich involved with European PROMETHEUS project [2].

Dickmanns applies a 4D approach exploiting spatio-temporal models of objects in the world to autonomous land and air vehicle control [1]. He has demonstrated autonomous state estimation for an aircraft in landing approach using a video camera, inertial gyros and an air velocity meter. Vision-based state estimation is also pursued at NASA Ames Research Center [15] using parallel implementation of multi-sensor range estimation for helicopter flight.

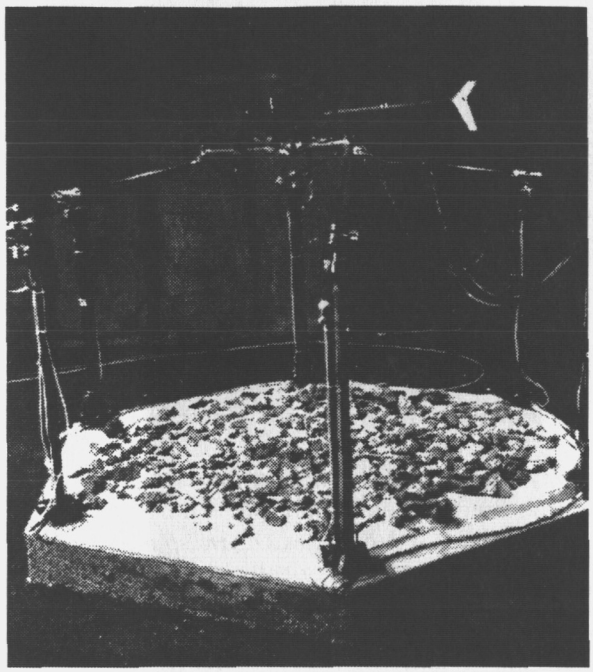


Figure 1: Indoor Testbed

An electrical model helicopter is supported by six light-weight graphite rods. A frictionless air bearing couples each rod with two-degree-of-freedom joints mounted on poles secured to the ground. Ground truth helicopter position is calculated from joint angles measured by shaft encoders.

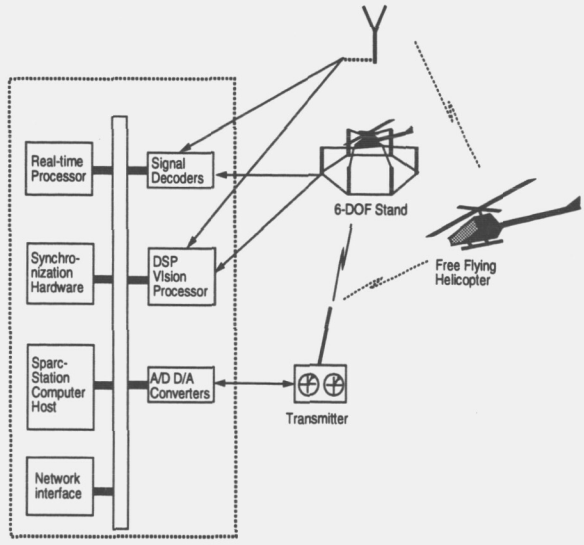


Figure 2: Testbed System Configuration

Indoor Helicopter Testbed

For practical, calibrated experimentation, we have designed and built an indoor helicopter testbed. It consists of an electrical model helicopter mounted on a 6-degree-of-freedom (6-DOF) test stand (see Figure 1). Using the testbed, we can test each critical component necessary for autonomous flight before attempting potentially dangerous outdoor free flight experiments.

Model helicopters provide an inexpensive, safe, and logistically manageable way to experiment with helicopter control. They are faithful reproductions of full size helicopters with respect to the crucial rotor controls and configurations. Control techniques developed for the model helicopters can be directly applied to larger scale helicopters.

The helicopter in our testbed is attached to a frictionless 6-DOF stand as shown in Figure 1. The stand provides ground truth measurement of the helicopter position and attitude, and also works as a safety device preventing crashes and out-of-control flight. The helicopter on the stand can fly freely in a cone-shaped volume six feet wide and five feet tall without major inertial variations from free flight. The helicopter is fastened to six fixed poles by six light-weight graphite rods. Each graphite rod is free to move through a frictionless air bearing mounted on a two-degree-of-freedom joint. The joint angles are measured by shaft encoders and used by the computer to calculate the helicopter's ground truth position and attitude for experiment evaluation.

The computer system configuration, shown in Figure 2, consists of a host computer, customized vision processor, a real-time processor, synchronization hardware, and interfacing equipment. A hand-held radio transmitter used by a model helicopter pilot is interfaced to real-time computers. Using this interface, we can send computer control signals to the helicopter. The same interface can be used for free flying helicopters.

With this testbed, we can perform controlled experiments over a wide range of conditions. We can create various wind conditions by using fans, terrain conditions by placing objects, and helicopter setups by adjusting the mechanisms. Because of the safety provided by the testbed, even potentially disastrous situations like the failure of critical helicopter parts can be tested.

Using a simplified helicopter dynamics model we have implemented a control system capable of hovering the helicopter using linear controllers tuned at different operating points. This control system provides us with a stable platform necessary for conducting low-speed and hovering experiments.

One apparent limitation of the test stand is its inability to support larger model helicopters capable of lifting several sensors at once. On the other hand, since the test stand provides ground truth data, we can simulate data from certain sensors by purposely corrupting

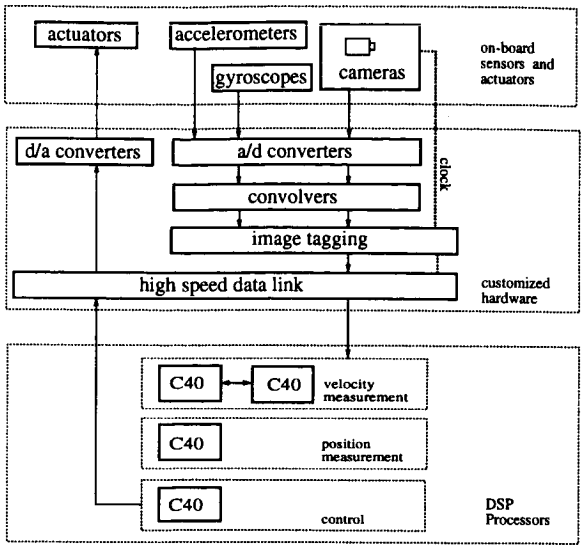


Figure 3: Vision Processor Structure

the stand data before using it. Different sensors can be individually characterized by comparing their response with ground truth data and their presence on-board the helicopter can be simulated during experiments.

Low Latency Vision Hardware for Helicopter Control

Our experience controlling model helicopters using the test stand has shown the necessity of velocity and position feedback rates of 15 to 30 Hz. Processing image data at these rates requires fast computers capable of acquiring and processing images at frame-rate (30 Hz). There are a number of new cost-effective compact CPU platforms designed for high speed data transfer and processing. Among the most popular are: SGS-Thomson inmos T9000 Transputer, Intel i860, and Texas Instruments TMS320C40 Digital Signal Processor (C40). Our development is based on the C40 platform primarily for its high speed communication ports each capable of transferring data at 20 MB/s. Other advantages include: programmable Direct Memory Access (DMA) well-suited for image windowing operations, flexible memory architecture and internal bus structure, and wide availability. The structure of our customized vision processor is shown by Figure 3.

We have achieved close integration of vision with other on-board sensors using customized hardware designed to interface with an array of C40 processors. This low-level integration is key in providing robust velocity and position estimation.

Digitizer Configuration

The helicopter has multiple on-board sensors: two ground-pointing black and white CCD video cameras, vertical and directional gyroscopes, and accelerometers for each translational axis. The data from these sensors is digitized using multiple special-purpose digitizers. In particular, our system provides variable sampling rates for image digitization. Typically, the NTSC video signal is sampled at 14.3 MHz which yields close to 1000 pixels per line. Conventional video digitizers choose 512 or 640 of these pixels per line during digitization. Since most CCD cameras have less than 1000 CCDs per line, we directly control digitizer sampling to reduce image data bandwidth and to provide more original image content. The aspect ratio of the image changes with sampling frequency and must be properly calibrated.

Convolution and Image Tagging

Fast convolution is essential for image preprocessing. In addition to edge detection and smoothing, matching and feature extraction can be performed using special convolution masks. We use real-time convolution hardware to perform Gaussian smoothing before processing images. To reduce image data bandwidth, we subsample the image using the digitizer before performing the smoothing operation. For the experiments described in this paper, 8×8 convolution masks were used on images sampled at 6 MHz pixel frequency.

Using similar convolution hardware, accelerometer and gyroscope data are sampled at 120 Hz and filtered by 64x1 Gaussian FIR filters. The filtered data is sampled and incorporated in the image data stream by an image tagger. Precise temporal matching of this data with the image is performed by using the camera's 60 Hz field vertical sync clock (VSYNC) and shutter speed. We use 1 millisecond shutter speed for tagging images accurately and reducing image blurring during helicopter motion.

High Speed Data Link

Because of the camera's VSYNC frequency, the processing time period for the sensor-tagged field images can only be multiples of 16.7 milliseconds. Barely missing an image due to long processing time is expensive since the processor must wait for a new image for proper synchronization. Image field digitization alone requires 16.7 milliseconds. During this time period the image must be transferred to the processor in order to achieve frame-rate (30 Hz) performance. We perform this transfer through a high speed data link designed to communicate with C40 processor comm-ports. This link incor-

porates small hardware buffers to convert the incoming synchronous image stream to the asynchronous comm-port protocol of the C40. In addition, since the image data is not directly entering a frame buffer, the high speed link provides proper comm-port synchronization with the camera using an internal state-machine. The comm-port design reduces CPU memory bus traffic by using C40's internal data buses and provides the ability to only transfer regions of interest using C40s versatile DMAs. These functions are crucial in improving processor speed.

Search Mission

As a concrete mission for an autonomous vision-guided helicopter, we envision a task of locating a known object in a predetermined outdoor area, for example, a particular car in a parking lot, and tracking the object by controlled helicopter flight.

The development of the indoor test stand allows us to conveniently simulate search mission scenarios using a variety of objects and terrain for visual tracking experiments. By carefully choosing these indoor experiments, we expect similar performance outdoors. The differences in flight altitude and terrain illumination can be resolved by small modifications to camera lenses, shutter speeds, and digitizing hardware.

Our mission is to search for a small car stranded somewhere in rough terrain. Performing this task requires object recognition to find the car, and visual measurement of position and velocity for autonomous flight. We have covered the stand base with gravel collected from the outdoor mission site to provide a realistic scene for our vision algorithms.

Velocity and Position Measurement

To measure helicopter velocity or position based on image data, we must first determine the displacement between consecutive images. This displacement in camera pixel coordinates is a function of camera attitude and distance relative to objects in the scene and camera calibration parameters such as focal length. For the indoor search experiments, camera attitude is estimated by gyroscopes and camera distance from the ground is estimated using the test stand. Performing outdoor experiments without the test stand requires altitude measurement by stereo vision possibly integrated with a laser rangefinder or microwave radar system.

The apparent displacement between consecutive images is a result of camera translation and rotation. Disambiguating rotation from translation is especially important for helicopter control since helicopter translation is directly a result of its change in attitude. Figure 4 shows the significance of this effect while the helicopter flares for reducing forward speed or stopping.

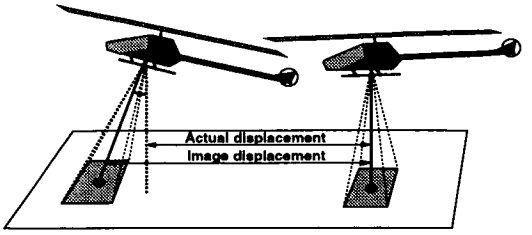


Figure 4: Effect of helicopter rotation

By carefully measuring the angular change between templates and images, we can estimate the effect of rotation and correct the image displacement to only reflect translational motion. This correction is useless without precise synchronization of gyroscope data with images. The drift common to all gyroscopes is not a problem here, since only the change in attitude is necessary from frame to frame.

Image Displacement Measurement

We use template matching to measure the displacement between consecutive images. We use sum-of-squared-differences (SSD) as our matching criteria. Each template is an $m \times n$ window of image intensities selected from the previous image. The best match of the template in the image can be determined by minimizing the SSD of the template and image pixels. To reduce the amount of computation, we restrict our search area to a small window around the template's neighboring pixels. The size of this search area is determined by helicopter altitude and anticipated worst case change in helicopter motion within one frame period. As the helicopter altitude decreases, the same translational motion causes a larger displacement in the image. The minimum altitude of the test stand is 1 meter and the on-board camera lens has 7.8 mm focal length. If we allow maximum helicopter velocity to be 1.5 meters per second during hover, our maximum image template displacement is 32 pixels per frame.

A coarse to fine strategy further improves search area and speed. We begin by using every fourth pixel to produce a coarse match for narrowing the search to 64 possible pixel locations. This estimate is improved to subpixel accuracy by fitting a parabolic surface to the SSD of the 64 match candidates. Figure 5 shows an example of a fitted parabola. A good parabola fit will refine the best single pixel match within ± 1 pixel. The parabola minimum is disregarded if it is not within one pixel of the single pixel match.

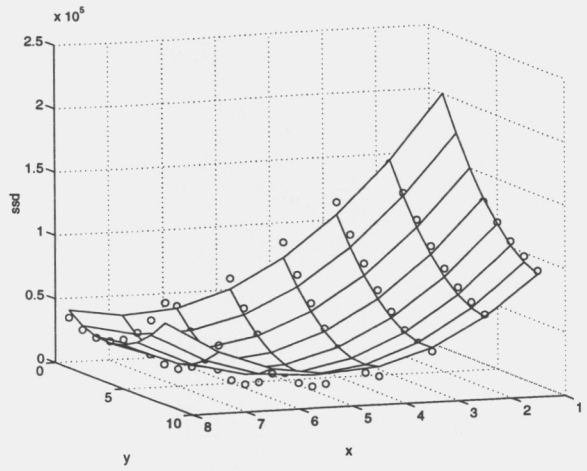


Figure 5: SSD Parabola Fit

In addition to subpixel accuracy, the fitted parabola provides match uncertainty information. A steep parabola versus a shallower one signals a more accurate match. Covariance matrices constructed from parabola coefficients will allow us to combine data from each template using a Kalman filter to produce the best estimate of image displacement.

For experiments reported here, we use four image templates for velocity and one template for position estimation as shown by Figure 6. The four velocity templates are 40×40 pixels in size and are positioned in each image quadrant. After each matching operation, the displacement of each template is calculated and the templates are updated with new image data from the same location.

Actual velocity measurement during flight is shown by Figure 7. This figure compares ground truth lateral and longitudinal velocity measurement (solid line) from the test stand with vision-based velocity estimates. The dashed and dotted lines in each figure represent vision-based velocity estimates with and without attitude correction. The correction was performed by measuring the attitude change between each template-image pair. Assuming images are taken from a locally flat surface, we can construct a transform, based on helicopter altitude and camera focal length, to convert the attitude change to a correction vector on the image plane for each template location. The effect of this correction is significant: 33 cm/s RMS error in lateral velocity measurement without attitude correction versus 5 cm/s after correction.

The position estimation template is 64×64 pixels in size and its location varies as the helicopter moves. This template is updated with image data from the best match in order to compensate for changes in helicopter

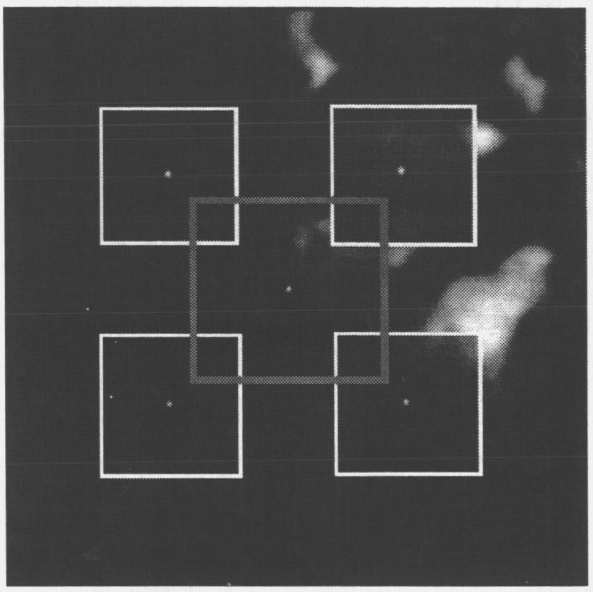


Figure 6: Image Templates

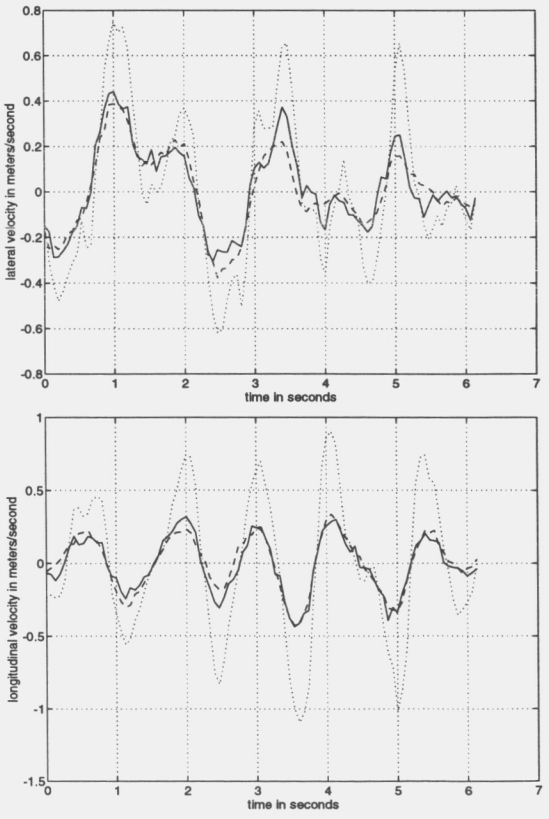


Figure 7: Vision-Based Velocity Measurement

The solid lines represent ground truth helicopter velocity from the test stand. The dotted lines show velocity based on image displacement alone and the dashed lines represent vision-based velocity with attitude correction.

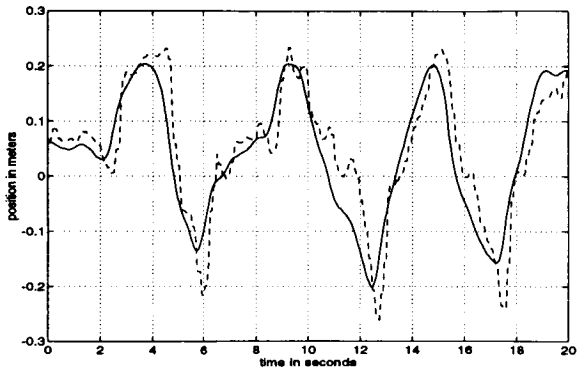


Figure 8: Position Measurement

The solid line represents ground truth helicopter position from the test stand. The dotted line shows position based on image displacement.

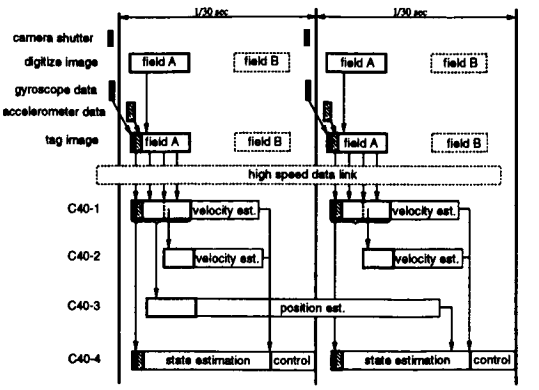


Figure 9: Data Flow and Synchronization

altitude and heading. If the best template match is close to leaving the camera view, the position template is loaded from the image center. A larger search area of 64 pixel displacement is used due to longer processing time. Figure 8 shows vision-based (dashed line) and ground truth (solid line) lateral position with respect to camera starting point. Attitude correction is more complicated in this case since the template changes position in the image plane. The figure shows uncorrected position estimation.

Position and Velocity Data Flow and Synchronization

We can not overemphasize the role of accurate synchronization in integration of on-board sensor data with high speed image processing. As observed above, attitude correction by synchronizing image and gyroscope data produces a significant improvement on position and velocity measurement accuracy. Figure 9 shows the data flow and synchronization we are performing for above helicopter motion estimation.

The solid vertical lines represent the camera VSYNC from the second image field (B). For high speed performance, only one image field (A) is used for motion estimation. The process begins with opening the camera shutter for 1 millisecond prior to VSYNC. Filtered gyroscope and accelerometer data is sampled with VSYNC and included in the image data stream by the image tagger. The tagged image is transferred to C40-1 which partitions field A for other C40s. The top half of field A is used by C40-1 and the bottom half by C40-2 for velocity estimation. In addition, field A is transferred to C40-3 for position estimation. Due to the high band-width of connections between C40s, it is possible to start image processing during image transfer. The transferring is performed by DMAs which do not interfere with data processing. C40-1 also transfers synchronized gyroscope and accelerometer data to C40-4 which is responsible for state estimation and control. The state estimation is performed by transforming image displacement data from other C40s to helicopter translational motion. The estimated translational velocity and position in conjunction with accelerometer and gyroscope data are used by linear control loops to control the helicopter.

Object Search

The vision-based velocity and position estimation provides the basic capability for hovering and low-speed flight necessary for our indoor search mission. Locating the object of interest is the next step. We use template matching to perform this search. A major difficulty in this approach is that object orientation is unknown. This requires templates of the object in all possible orientation for matching. Methods such as K-L expansion [18] can be used to reduce computational complexity and storage of necessary templates. Another problem stems from varying helicopter altitude which will change the size of the object in the image. Close regulation and measurement of helicopter altitude is necessary to further reduce the complexity of the search.

We are conducting the search using a set of twenty 32×32 templates. These twenty templates, generated by K-L transform techniques, are sufficient for locating objects with $\pm 40°$ orientation discrepancy as accurately as one degree resolution. The processing frequency for searching the entire image is 3 Hz using one C40 processor. Upon locating the object, the position estimator can now use the object in the image as its template providing relative helicopter position necessary for object tracking.

Conclusions

We have successfully developed the key components necessary for vision-guided autonomous flight. As our experimental results demonstrate, we are achieving real-time low latency image processing at suitable rates to stably fly helicopters. The major elements in our development have been custom designed vision hardware and indoor testbed. In addition to high speed processing, customized hardware provides flexible integration of on-board sensors which significantly improves vision-based state estimation. The indoor testbed provides convenient calibrated experimentation which is essential in building real autonomous systems.

References

[1] E.D. Dickmanns. A general dynamic vision architecture for UGV and UAV. *Journal of Applied Intelligence*, 2:251–270, 1992.

[2] E.D. Dickmanns and B.D. Mysliwetz. Recursive 3-D road and relative ego-state recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(2):199–213, Feb. 1992.

[3] David R. Downing and Wayne H. Bryant. Flight test of a digital controller used in a helicopter autoland system. *Automatica*, 23(3):295–300, 1987.

[4] H. Ekerol and D.C. Hodgson. A machine vision system for high speed object tracking using a moments algorithm. *Mechatronics*, 2(6):555–565, Dec. 1992.

[5] Dale F. Enns. Multivariable flight control for an attack helicopter. *IEEE Control Systems Magazine*, April 1987.

[6] C. Harris and C. Stennett. Rapid - a video rate object tracker. *BMVC90 Proceedings of the British Machine Vision Conference*, pages 73–77, Sept. 1990.

[7] R. A. Hess and K. K. Chane. Preview control pilot model for near-earth maneuvering helicopter flight. *Journal of Guidance*, 11(2):146–152, April 1988.

[8] Ronald A. Hess and Yoon C. Jung. An application of generalized predictive control to rotorcraft terrain-following flight. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5), 1989.

[9] H. Inoue, T. Tachikawa, and M. Inaba. Robot vision system with a correlation chip for real-time tracking, optical flow and depth map generation. *Proc. 1992 IEEE Int. Conf. on Robotics and Automation*, 2:1621–1626, May 1992.

[10] Jurgen Kaletka and Wolfgang von Grunhagen. Identification of mathematical derivative models for the design of a model following control system. *Vertica*, 13(3):213–228, 1989.

[11] R.C. Luo and R.E. Mullen Jr. Combined vision / ultrasonics for multi-dimensional robotic tracking. *Sensor Fusion: Spatial Reasoning and Scene Interpretation; Proceedings of the SPIE*, 1003:113–122, 1989.

[12] N.P. Papanikolopoulos, P.K. Khosla, and T. Kanade. Visual tracking of a moving target by a camera mounted on a robot : a combination of control and vision. *IEEE Trans. Robot. Autom.*, 9(1):14–35, Feb. 1993.

[13] A.A. Rizzi, L.L. Whitcomb, and D.E. Koditschek. Distributed real-time control of a spatial robot juggler. *Computer*, 25(5):12–24, May 1992.

[14] M. Sugeno, T. Murofushi, J. Nishino, and H. Miwa. Helicopter flight control based on fuzzy logic. *Fuzzy Engineering toward Human Friendly Systems*, 1991.

[15] R.E. Suorsa and B. Sridhar. A parallel implementation of a multisensor feature-based range-estimation method. *Proc. 1993 IEEE Conference on Computer Vision and Pattern Recognition*, pages 379–385, 1993.

[16] Charles E. Thorpe. *Vision and Navigation: the The Carnegie Mellon Navlab.* Kluwer Academic Publishers, 1990.

[17] Barbara K. Townsend. The application of quadratic optimal cooperative control synthesis to a CH-47 helicopter. *Journal of the American Helicopter Society*, pages 33–44, January 1987.

[18] M.A. Turk and A.P. Pentland. Face recognition using eigenfaces. *Proc. 1991 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 586–591, June 1991.

[19] A. Yue, I. Postlethwaite, and G. Padfield. H^∞ design and the improvement of helicopter handling qualities. *Vertica*, 13(2):119–132, 1989.

REAL-TIME TRACKING OF OBJECTS FOR SPACE APPLICATIONS USING A LASER RANGE SCANNER*

F. Blais, R.A. Couvillon, M. Rioux
Institute for Information Technology
National Research Council of Canada
Ottawa, Ontario, Canada K1A 0R6

S.G. MacLean
Canadian Astronaut Program
Canadian Space Agency
St-Hubert, Québec, Canada J3Y 8Y9

Abstract

Real-time tracking of multiple targets or geometrical features of an object using a variable resolution laser scanner is presented. The scanner is characterized by its robustness to ambiant illumination, even the sun shining directly into the sensor, and its tracking resolution. The sensor to extract registered range and intensity information for each scanned point on the object at a rate of 18 kHz uses two high-speed galvanometers and a collimated laser beam. Three-dimensional real-time tracking using Lissajous patterns proves to be very attractive for space applications. Integration with the existing photogrammetry-based Advanced Space Vision System (ASVS) is discussed.

Introduction

It is now well accepted that vision will play a major role in both supervised and unsupervised operations for the automation of several space-related activities. Specifications state that the Artificial Vision Unit will support rendez-vous and proximity operations including payload tracking, capture, and berthing in both teleoperation and adaptive control modes.[1,2] It is also recognized that vision will play a major role during the assembly and maintenance operations of the space station.

The current Artificial Vision Function (AVF) is based on the Space Vision System (SVS) that was demonstrated on CANEX-2. The SVS analyzes video signals from the closed circuit television system of the space shuttle and provides real-time position and orientation information about an object with a cooperative target array.[2] The baseline requirements for the Artificial Vision Function are as follows:

- To identify a suitably illuminated object (target) from its video image.
- To estimate the position, attitude, translation, and rotational rate of the object.
- To provide appropriate camera control to track the object.
- To be able to track objects before capture by manipulators and berth objects/payloads handled by manipulators.

To achieve robust adaptive control, supervision, and inspection, the vision system must be 100% operational throughout the changing illumination conditions in orbit. Unfortunately, the quality of the images produced by standard video-camera-based systems is adversely affected by the presence of the sun or any other strong source of light. Poor contrast between features on the object and background, and saturation of the photo-detector array often make it difficult to analyze the video images. Video camera reliability during normal operation is questionable and can compromise the success of the operation or at least seriously limit its practical use.

Although camera-based systems are very attractive because of their ease of use and simplicity of integration with existing equipment, it is highly desirable to offer a complementary vision system that will not be restricted by operational conditions such

* NRC 37109

Copyright c 1994 by Francois Blais. Published by the American Inst: ute of Aeronautics and Astronautics, Inc. with permission.

as sun interference. The proposed laser scanner approach offers the advantage of being almost 100% operational throughout the changing illumination conditions in orbit. The technique, developed at NRC in collaboration with the CSA, is designed to be insensitive to background illumination such as the earth albedo and the sun radiation and most of its reflections.[3,4] Immunity to background interference is the result of the very narrow band of frequencies emitted by the laser light that can be tuned by optical filters, by special optical design to extract range information, and by proprietary real-time signal processing techniques used to distinguish between the laser beam and any remaining sources of interference (e.g., specular reflections).

The Laser Scanner

Figure 1 shows the geometry of this prototype based on the auto-synchronized time-flying single-point technique.[5] The system is comprised of two orthogonally mounted scanning mirrors: the X-axis scanning mirror, used for range triangulation, and the Y-axis mirror. The two mirrors are driven by accurate galvanometers. The light beam generated by the laser is deflected by a mirror and scanned on the object. A camera, consisting of a lens and a position-sensitive photodetector, measures the location of the image illuminated point on the object. By simple trigonometry, the three-dimensional coordinates of that point are calculated.[6,7]

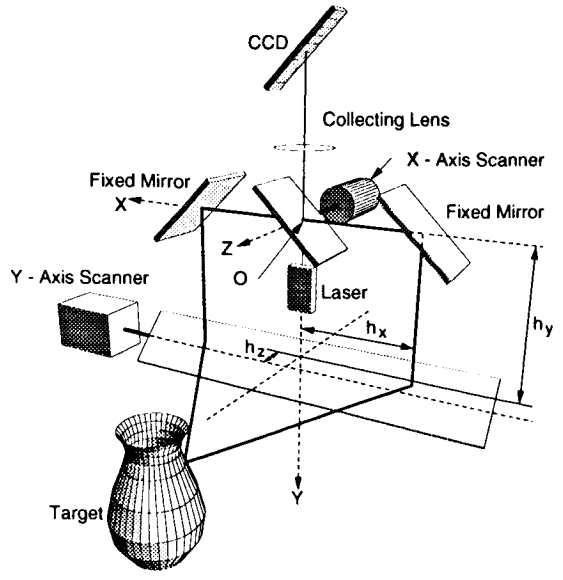


Figure 1: Schematic representation of the dual-axis auto-synchronized range scanner (reprinted from Reference 7).

Some of the advantages of the auto-synchronized technique are a large field of view, high accuracy, and immunity to ambiant illumination.[8] A conventional camera must continuously monitor the whole field of view, making it very susceptible to sun interference. The laser scanner technique has a small instantaneous field of view limiting the undesired interference. Such a system is optically light efficient because the received laser power is focused onto a small spot thereby increasing the signal-to-noise ratio of the returned signal. Automatic control of the laser power source further extends the dynamic range of the scanner.

Figure 2 illustrates the equivalent optical geometry. The basic concept is that the projection of the light spot is synchronized with its detection. The instantaneous field of view of the position sensor follows the spot as it scans the scene. An external optical perturbation can interfere with the detection only when it intercepts the instantaneous field of view of the scanner. At this level, electronic signal processing is used to filter these false readings to obtain the correct 3-D measurement. The total field of view of the laser camera is related only to the scanning angles of the galvanometers and mirrors as opposed to a conventional camera where field of view and image resolution are intimately linked, the larger field of view the smaller pixel resolution achievable.

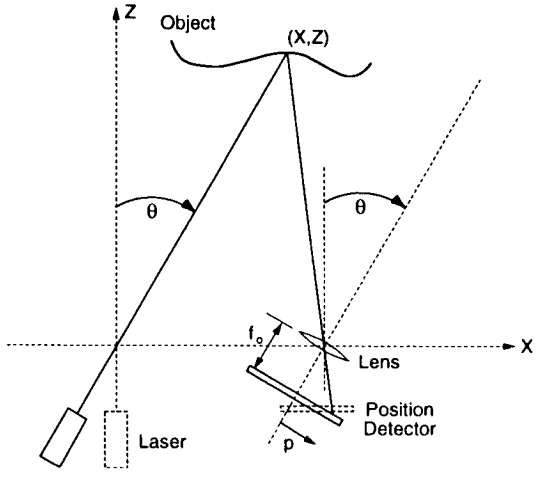


Figure 2: Basic principle of synchronized scanner, second axis not shown (reprinted from Reference 8).

Although laser scanners are usually slower than their TV camera counterparts when used in a conventional imaging or raster scan mode of operation, the same laser scanner can obtain refresh rates of more than 130 Hz with a pointing resolution of 15000 elements × 15000 elements in the tracking mode (single target). Consequently, the resolution and speed of the laser scanner exceed conventional video cameras. The high pointing accuracy of a laser beam and the large depth of field of the scanner enable the acquisition of large 3-D images of 4000 pixels × 4000 pixels or more. To achieve similar resolution, a video camera would need a high-quality optical lens and specially designed high-resolution CCD cameras.

A custom-designed galvanometer controller board gives access to any pixel in the field of view of the range sensor and provides the random access tracking capability.[3,4] The controller permits interrogation of any point in the scene without having to sample the entire field of view of the scanner. This Region Of Interest (ROI) sampling technique offers great potential for rapid and efficient acquisition of dense and accurate 3-D images over a large volume of measurement.

Figure 3 is a photograph of the laboratory prototype of the auto-synchronized laser scanner developed for this application. Tables I and II and Figures 4 and 5 summarize the characteristics of this laser scanner prototype in the triangulation mode of operation.

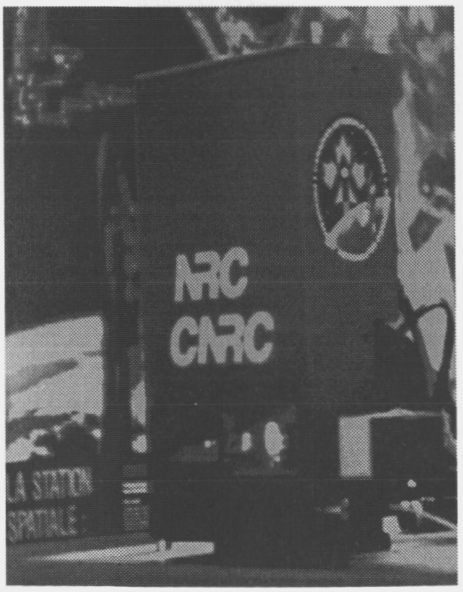


Figure 3: Photograph of the laser scanner prototype.

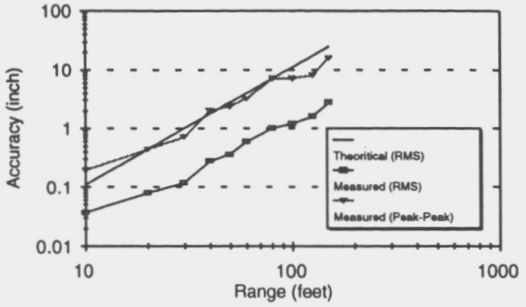


Figure 4: Range resolution (single point).

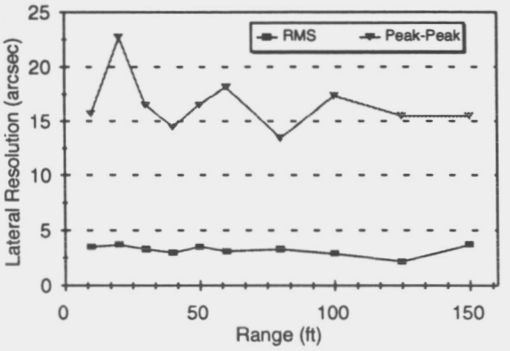


Figure 5: Angular pointing resolution (tracking mode using Lissajous figures).

Real-time Tracking

Real-time tracking of targets or geometrical features on an object is implemented using Lissajous figures, to obtain good scanning speed and accuracy[3] A Lissajous figure is mathematically defined by

$$\theta(t) = A_\theta \cos(m\omega t + \delta_\theta) + \theta_0 \quad (1)$$

$$\phi(t) = A_\phi \cos(n\omega t + \delta_\phi) + \phi_0 \quad (2)$$

where A_θ and A_ϕ are the amplitudes of the Lissajous

Table I: Physical characteristics.

Dimensions	10.5 in × 6.25 in × 4 in
Data Type	3-D and Intensity
Field of view	30° × 30°
Working range	0.6 ft to TBD (>150 ft)
Range accuracy	See Figure 4
Acquisition Speed	18 kHz
Scanning System	Galvanometers
Resolution X-Y	1 / 15000

pattern, ω the refresh rate of the scan, δ_θ and δ_ϕ the relative phases of the sinewaves, t the sampling time, and θ_0 and ϕ_0 the position of the center Lissajous pattern. The pattern shape is defined using parameters m:n. For example the pattern illustrated in Figures 6 and 7 is a 3:2 Lissajous figure.

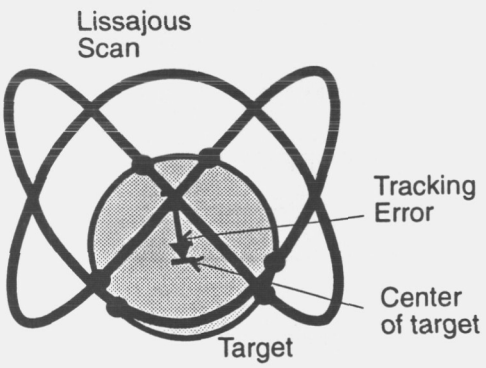


Figure 6: Tracking of a circular target using a 3:2 Lissajous pattern.

Lissajous patterns are used to scan objects at refresh rates exceeding the frequency response of the mechanical deflection system. The scanning device is not required to stop the acquisition after each trace line as opposed to the raster-type scan illustrated in Figure 8. Instead, the full pattern is used to acquire both range and intensity information. Furthermore, the natural inertia of the galvanometer-mirror structure smooths the scanning pattern and hence increases the pointing accuracy of the tracking

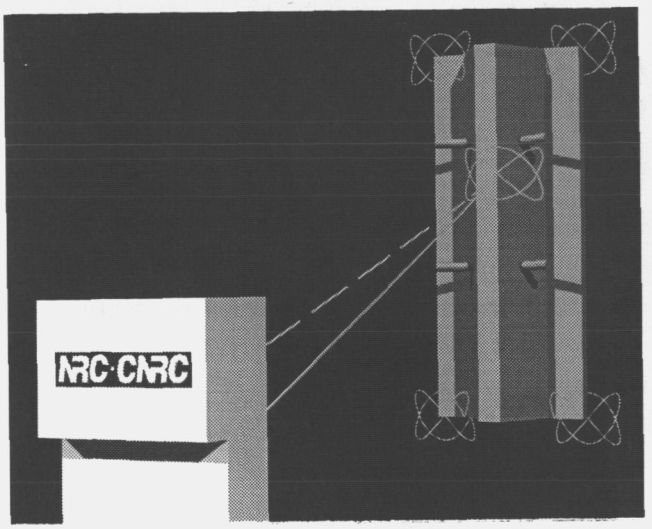


Figure 7: Real-time tracking of geometrical targets (e.g., corner or targets) using Lissajous patterns and the laser range sensor.

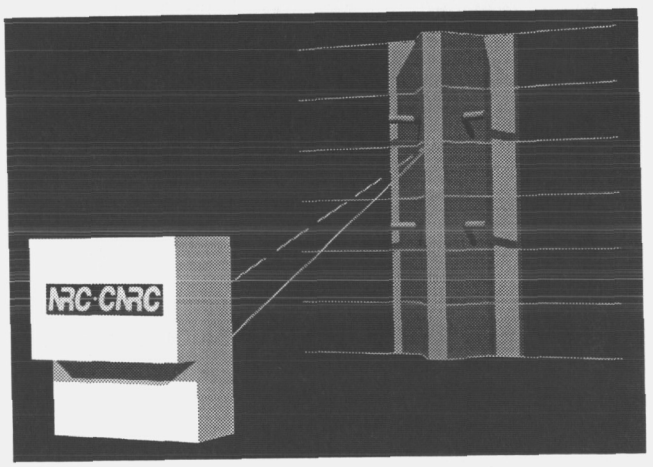


Figure 8: Conventional raster-type acquisition of the 3-D shape of the object.

system. As well, Lissajous pattern signals are optimally filtered using the Fourier transform (or notch filter), thus reducing electrical noise, quantization effects, and distortions.[9]

Tracking is implemented using the 3-D range information on the Lissajous pattern, the returned intensity signal from the object, or a combination of both. For example, the location and orientation of the geometrical target is obtained with either the 3-D or intensity edges of the object or the intersection of the measured surfaces on an object.

Table II: Performances characteristics of the laser scanner.

Laser Scanner	Video Cameras
General	
- Excellent immunity to ambiant illumination - High-resolution images and large depth of view - 3-D and registered intensity images - New technology not as widely used	- Highly susceptible to ambiant illumination - Low hardware cost - Use on-board closed-circuit video cameras system
Optics	
- Optically efficient, excellent signal-to-noise ratio because all the laser energy is concentrated in a single point - Simple on-axis optics - Large depth of view (max/min range) and small instantaneous field of view - Optical magnification produces good range resolution and large images of the sun to simplify the signal processing and reduce saturation - Laser light easily filtered using narrow-bandwith optical interference filters - Eye safety requirements for terrestrial applications (eye-safe if laser at 1.5 μm is used)	- Simple camera configuration - Work typically under ambiant illumination or projectors - Adversely affected by ambiant illumination and sun interference - Large instantaneous field of view makes the system susceptible to saturation - No optical filtering possible because of the use of ambiant illumination (if laser projectors are used large-bandwith optical filters are required) - Compromise between optical magnification and field of view (focal length) and sensitivity and depth of view (f-number)
Scanning devices (galvanometers)	
- Large scanning angles, high pointing resolution - Randomly addressable, almost any scanning patterns can be programmed - Mechanically moving devices (relatively slow) - Temperature sensitive*	- No moving parts except if rotation stage is used for camera pointing - Mechanically rugged systems - Temperature sensitivity is reduced by years of design and mature technology
Electronics	
- Simplify the processing of the CCD image - Further improve the immunity to ambiant illumination - Modular architecture easily upgradable - Random control and real-time implementation requires good knowledge of the dynamics of the sensor and the application**	- 2-D image processing required at video rate - Special technique still to be developed to reduce the effect of changing illumination on the targets - Modular low-cost electronics - All solid-state

* Temperature compensation is implemented using synchronization photo-detectors.[8]

** This is valid for any accurate measurement system.

Figure 6 illustrates the tracking principle using a Lissajous figure. The target can be a cylinder, a sphere, or simply a colored target . For long-range measurement a retroreflective target (corner cube or 3M retroreflective tape) is used. Assuming that $I(t)$ and $Z(t)$ are, respectively, the intensity and range measurements for each point of the Lissajous pattern defined by equations 1 and 2, the tracking error on the target is computed from the centroid measurements weighted by the intensity of the returned laser signal:

$$I_{val}(t) = I(t) - I_{ref} \quad \text{if } (I(t) > I_{ref}) \text{ and } (\Delta r_{min} < (r(t) - r_0) < \Delta r_{max}) \quad = 0 \quad \text{otherwise} \tag{3}$$

$$\Delta r = \frac{\sum r(t)\, I_{val}(t)}{\sum I_{val}(t)} - r_0 \tag{4}$$

$$\Delta\theta = \frac{\sum \theta(t)\, I_{val}(t)}{\sum I_{val}(t)} - \theta_0 \tag{5}$$

$$\Delta\phi = \frac{\sum \phi(t)\, I_{val}(t)}{\sum I_{val}(t)} - \phi_0 \tag{6}$$

If a target is lost then

$$\Delta r = \Delta\theta = \Delta\phi = 0 \quad \text{if } \left(\sum I_{val}(t) = 0\right) \tag{7}$$

θ_0, ϕ_0, r_0 is the position of the scan and consequently the expected position of the target. The intensity threshold I_{ref} and range validation window r_{min} and r_{max} are specific for a target type. Any point outside the range/intensity window is considered not valid and disregarded. Range is extremely useful in equation 3 to make the technique very robust to external perturbations (e.g., $r=\infty$ for the sun)

Tracking is implemented in the spherical coordinate system (r,θ,ϕ) of the laser scanner using the digital feedback structure defined, using the z-transform by:

$$\vec{\delta} = H_{dir} * \Delta\vec{\delta} + H_{pred} * \vec{\delta}_{pred} + H_{adj} * \Delta\vec{\delta}_{adj} \tag{8}$$

$$\vec{\delta} = [r, \theta, \phi]$$

$$H_{dir} = \frac{\alpha_0 + \alpha_1 z^{-1} + \alpha_2 z^{-2}}{1 - \beta_1 z^{-1} - \beta_2 z^{-2}} \tag{9}$$

$$H_{pred} = \frac{\gamma}{1 - (1-\gamma) z^{-1}} \tag{10}$$

$$H_{adj}(z) = \frac{1}{1 - \zeta z^{-1}}, \quad \zeta < 1 \tag{11}$$

z^{-1} is the z-transform delay of one sample. H_{dir} is the direct feedback loop internal to the laser scanner, H_{pred} is the predicted or expected position of the target, and H_{adj} is an external user control adjustement. H_{dir} implements a classical deadbeat controller reducing the tracking error to zero such that the Lissajous pattern is always centered on the target. The predicted or expected position H_{pred} is used to give an estimate of the position of the target. This estimate is obtained either from the known or calculated position of the object or from the ASVS-laser scanner interface. H_{adj} allows an immediate fine adjustment of the target position. This is especially useful in the search mode when the target is not found (equation 7) and the predicted position is not accurate enough (e.g., during initial search) to lock the laser scanner pattern on the target. External user control is used to move the pattern over the target. Equation 11, a lossy integrator, gradually removes this external correction and is replaced by the correction H_{dir}.

The laser scanner automatically adjusts the size of the scan according to the distance of the target from the camera:

$$A_\theta = A_\phi = \frac{A_{ref}}{r} \tag{12}$$

For each target, a set of parameters A_{ref}, I_{ref}, Δr_{min},Δr_{max} is defined. Optimum coverage of the surface of the target and pointing resolution are obtained because the Lissajous pattern is always centered and scaled on the target independently of the distance of the target.

Multiple Tracking of Targets, Photogrammetry, and the Laser Scanner

The photogrammetry mode of operation of the ASVS to be used on board the space shuttle[1] and the laser scanner are compatible and easily interfaced; the deflection angles of the galvanometers directly provide the angular separation between the targets. Then, with photogrammetry techniques, the object position X-Y-Z and attitude parameters yaw-pitch-rotation are computed by minimizing the quadratic error between the expected position of the targets computed from the model of the object and the measured positions provided by the laser scanner.[2]

Real-time and physical constraints caused by switching quickly between multiple targets and limited by the inertia of moving mechanical devices must be considered, based on the application requirements. A careful investigation of the complete mechanical, optical, and electrical characteristics of the hardware and software is needed to achieve optimum tracking performance.

The laser scanner is programmed to sequentially scan different sections or targets on one or multiple objects, as illustrated in Figures 7 and 9. A list of all possible scanning strategies used for a given task are pre-defined. Each strategy contains a list of the visible targets to scan and is optimized for a given application. The laser scanner uses this list to sequentially scan the targets on the different objects.

A scenario under study for the assembly of the space station is the berthing of the LAB/A Module on the APN (Aft Port Node) CBM interface in the Space Station using the Remote Manipulator System (RMS) as graphically illustrated in Figure 9. The vision system task is to provide the position and orientation of the LAB/A module relative to the APN and ITA (Integrated Trust Assembly, not shown) modules as well as visual guidance during berthing operations.

The visible and nonvisible targets on the two main objects are:

Targets	*Module*
T_1 to T_8	LAB/A
T_9 to T_{16}	APN (Aft Port Node)

Only the visible targets T_1 to T_4 and T_9 to T_{12} are shown in Figure 9.

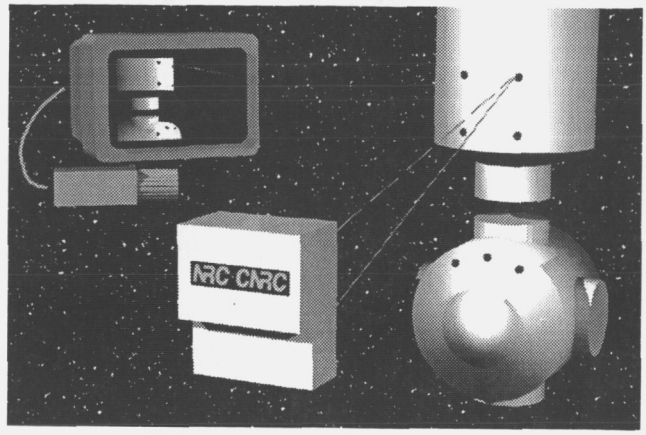


Figure 9: Example of tracking for the assembly of the space station.

The following strategies are defined for this visible section of the objects:

Strategy	=	*Scanning Target List*
S_0	=	T_1, T_2, T_3, T_4
S_1	=	$T_9, T_{10}, T_{11}, T_{12}$
S_2	=	$S_0, T_9, S_0, T_{10}, S_0, T_{11}, S_0, T_{12}$

It is important to acquire fast moving targets as often as possible because of the sequential nature of the scanning method. The basic scanning strategies S_0 and S_1 scan only the targets on their respective objects. They are used to initialy locate a relatively fast moving object. When the object is locked by the scanner, the scanning strategy is switched to S_2 to measure the relative position of the LAB/A module with respect to the whole structure. In this example the APN module is considered to be almost stationary and therefore S_2 instructs the laser scanner to scan the moving LAB/A module more often than the other objects. All the targets from the LAB/A module are scanned (S_0) followed by one target from the APN module. Although only three strategies are shown, any number can be defined, depending on the application.

The approach used for object tracking allows the operator to fully define the tasks required by the tracking system. Instead of having a single "target or object" detection algorithm controlling the scanner, the user can dynamically tune or replace any element for a given application. New modules are easily built to improve system performances.

The tracking refresh rate for the LAB/A module is currently limited to 15 Hz for this experiment. Geometrical detection and tracking are computationaly expensive and need more processing power to achieve faster tracking speed than the current single 68020 VME processor can provide. Integration of parallel processing using multiple TMS320C40 DSP is planned. Furthermore, several key elements of this system, including geometrical target localization algorithms, absolute real-time calibration (vs. relative calibration), and optimized corrector-predictor tracking techniques, are only at their preliminary stage of study. As the system improves in performance, further modifications can be introduced without redesigning the whole software or hardware environment.

Laser Scanner and Three-dimensional Imagery

A more conventional imagery method can also be used for tracking, at lower speed than the Lissajous technique. Three-dimensional imagery using a laser scanner has the major advantage of creating high resolution images of the object under inspection. Images of more than 4000 points x 4000 points are easily acquired Figure 8 illustrates the raster-type acquisition mode, similar to conventional video cameras except that 3-D range information and intensity, in perfect registration, are obtained for all the points scanned in the image. Resolution is also higher than with conventional CCD cameras, as discussed previously. It is usually assumed that an object is relatively stationary with respect to the scanner.

Figure 10 shows a 3-D raster image of a copy of the CTA testing module used during testing of the SVS system on board Space Shuttle Flight STS-52. It is used here for evaluation of geometrical tracking algorithms. Other scanning methods have also been proposed based on the random access laser scanner discussed here.[10]

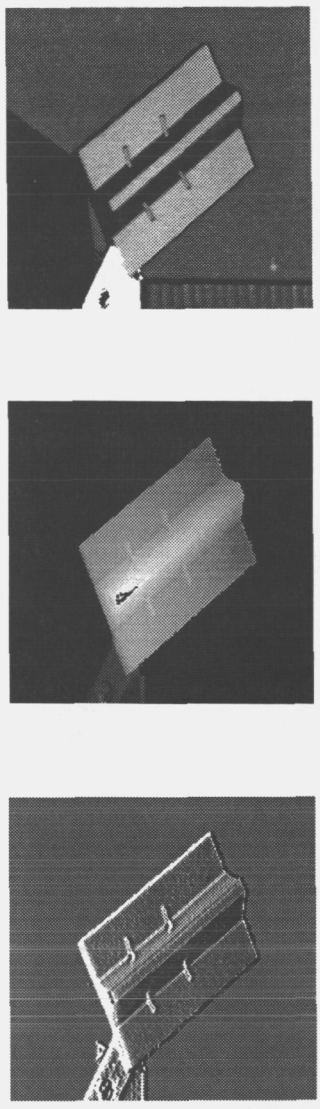


Figure 10: Raster type image of a model of the CTA used during space shuttle flight STS-52: (top) intensity image, (middle) range image encoded in grey levels , (bottom) pseudo shaded (derivative) of the range image (texture).

Conclusion

Real-time tracking of targets on an object and acquisition of high-density three-dimensional images have been demonstrated using a random access 3-D laser scanner. The prototype has the potential of automatically tracking, in three dimensions, either geometrical features of the object itself or targets attached to it. Excellent immunity to ambient illumination and sun interference is obtained.

Real-time tracking of targets on an object is realized using Lissajous-type scanning patterns or raster type images, depending on the speed of the moving obje ct. The Lissajous scanning figures give high pointing resolution, excellent stability, and good object position refresh rate.

Tracking error feedback is simultaneously obtained from three different sources: (1) direct tracking error based on each individual targets, (2) global predicted target position based on the calculated estimate of the object position, and (3) external feedback control from the human operator. Tracking of multiple targets using Lissajous patterns is based on user defined scanning strategies pre-programmed in the laser scanner according to the application requirements.

Angular tracking resolution of 20 arcsec RMS at 150 ft, for a field of view of 30° was measured (1/15000). A more complete study of the calibration parameters and of temperature variations on the laser scanner is still required to fully characterize the exact performance of the sensor. To obtain an absolute accuracy measure for the full working volume (160 00 yd^3), the difficult problem of obtaining a reference system more accurate than the laser scanner must be solved. Only resolution is quoted for the moment. Full calibration of the laser scanner/ASVS system combination is currently in progress.

References

[1] S.G. MacLean, M. Rioux, F. Blais, J. Grodski, P. Milgram, H.F.L. Pinkney, B.A. Aikenhead, "Vision System Development in a Space Simulation Laboratory", in Close-Range Photogrammetry Meets Machine Vision, *Proc. SPIE 1394*, 8-15 (1990).

[2] H.F.L. Pinkney, "Theory and Development of an on-line 30 Hz Video Photogrammery System for Real-time 3-Dimensional Control", *Proceedings of the International Society of Photogrammetry Symposium on Photogrammery Industry*, Stockholm, 1978.

[3] F. Blais, M. Rioux, S.G. MacLean, "Intelligent, Variable Resolution Laser Scanner for the Space Vision System", in Acquisition, Tracking, and Pointing V, *Proc. SPIE 1482*, 473-479 (1991).

[4] F. Blais, J.-A. Beraldin, M. Rioux, R.A. Couvillon, S.G. MacLean, "Development of a Real-time Tracking Laser Range Scanner for Space Applications", *Proceedings Workshop on Computer Vision for Space Applications*, Antibes, France, 1993.

[5] M. Rioux, "Laser Range Finder Based on Synchronized Scanners", *Appl. Opt.*, 23(21), 3837-3844 (1984).

[6] J.-A. Beraldin, M. Rioux, F. Blais, G. Godin, R. Baribeau, "Model-based Calibration of a Range Camera", *Proceedings IAPR*, The Hague, Netherlands, 1992.

[7] J.A. Beraldin, S.F. El-Hakin, L. Cournoyer, "Pratical Range Camera Calibration", in Videometrics II, *Proc. SPIE 2067*, 21-31 (1993).

[8] F. Blais, M. Rioux, J.A. Beraldin, "Practical Considerations for a Design of a High Precision 3-D Laser Scanner System", in Optomechanical and Electro-Optical Design of Industrial Systems, *Proc. SPIE 959*, 225-246 (1988).

[9] S.J. Orfanadis, "Optimum Signal Processing: An Introduction", *Macmillan Publishing Company, 2nd Edition*, 1985.

[10] Liu Y., Laurendeau D., "Processing of a Multi-scale Triangulated Surface Model of a 3-D Scene for a Robotics Application", in Geometric Methods in Computer Vision II, *Proc. SPIE 2031*, 392-403 (1993).

AIAA-94-1242-CP

INTEGRATION FOR NAVIGATION ON THE UMASS MOBILE PERCEPTION LAB

Bruce Draper**, Claude Fennema†, Benny Rochwerger+
Edward Riseman*, Allen Hanson*

Abstract

The Mobile Perception Laboratory (MPL) is an autonomous vehicle designed for testing visually guided behaviors, such as road following, passive obstacle detection, landmark-based navigation and model acquisition [Hanson, Riseman, & Weems 93]. The research is being conducted under the sponsorship of the ARPA Unmanned Ground Vehicle (UGV) Program in the Computer Vision Laboratory at the University of Massachusetts.

The focus of this paper is on integrating multiple behaviors into a single, coherent system. It presents the ISR3, a new tool for interprocess communication, the storage and retrieval of transient, image-based data, and the long-term management of 3D data. It also presents the script monitor, a process control mechanism for invoking, monitoring and destroying concurrent sets of visual behaviors in response to dynamic events and the systems stated goals.

1. The Mobile Perception Lab (MPL)

The experimental laboratory vehicle for this effort is the UMass Mobile Perception Lab (MPL), a heavily modified Army HMMWV ambulance (Figure 1) that is equipped with actuators and encoders for the throttle, steering and brake. The interface to the on-board computer system is through a 68030-based controller board. Electrical power is provided by an on-board 10kW diesel generator feeding uninteruptable power supplies and conditioners. The MPL closely matches CMU's NavLab II, with modifications and component installation performed by RedZone, Inc., a Pittsburgh-based firm specializing in custom robotics.

Figure 1. The Mobile Perception Laboratory

The vehicle's sensor package includes a Staget, which is a stabilized platform capable of rotating a full 360^{o}. The Staget is mounted at the center of the cab roof and contains a CCD color camera and a FLIR sensor in a weatherproof enclosure. Two forward-looking stereo cameras and a forward looking color CCD camera are mounted in a rectangular enclosure at the front edge of the cab's roof. The primary computing engine for vision processing, goal-oriented reasoning and path planning is a Silicon Graphics 340GX four-node multiprocessor. The multiprocessor is interfaced to the sensor suite through a Datacube MaxVideo20 processor, which provides frame rate image processing for certain types of operators. Space and power has been provided for the possible addition in the future of a 16K Image Understanding Architecture (IUA) [Weems 1993], a massively parallel heterogeneous processor.

*Prof., Computer Science, Univ. of Massachusetts
**Sr. Postdoctoral Researcher, Computer Science, Univ. of Massachusetts
†Asst. Prof., Computer Science, Mt. Holyoke College, So. Hadley, MA
+Graduate student, Computer Science, Univ. of Massachusetts

Copyright © 1993 by the authors. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

The physical lay-out of equipment on the vehicle is depicted in Figure 2. The first programmer station is located in the HMMWV's passenger seat, with a 17" color X-terminal fixed to the metal platform between the passenger's and driver's seats. The second programmer station is located behind and slightly above the driver, and includes a car seat, mounting brackets for both an SGI color terminal and a small SONY monitor for viewing

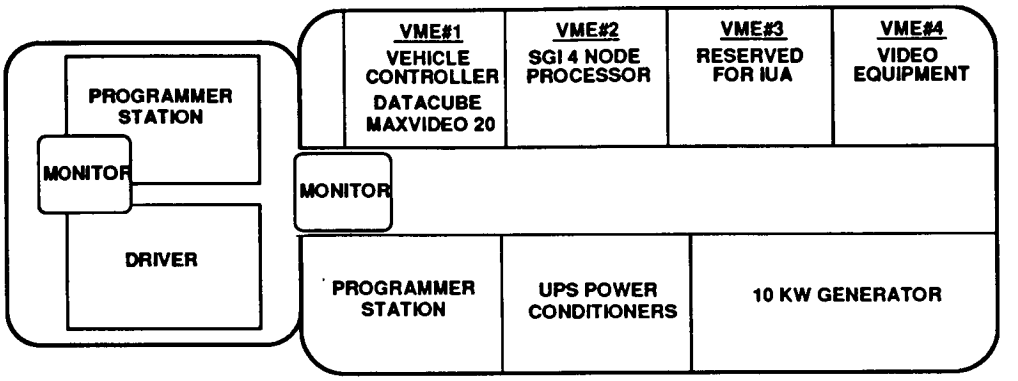


Figure 2. Interior layout of the MPL.

raw TV signals. Behind this programmer station is the diesel generator and power conditioning systems. The passenger's side rear hold four enclosed, air conditioned 19" computer frames for the on-board computer systems. The first frame holds the vehicle controller, image digitizers and frame stores, the MaxVideo20, and the Staget controller and interface. The second computing frame contains the Silicon Graphics multiprocessor, disk drives, power supply and (removable) tape drive. The third frame is reserved for the Image Understanding Architecture (IUA). The fourth frame contains video recorders for collecting experimental data.

2. Integration of Vision Modules for Real-Time Control: An Overview

MPL is an experimental laboratory for testing and integrating different approaches to problems in autonomous navigation, including, but not limited to, landmark-based navigation, obstacle detection and avoidance, model acquisition and extension , road following, and path planning. It is therefore important that MPL have a software environment where multiple visual modules, addressing different subtasks, can be easily integrated, and where researchers can quickly experiment with different combinations and parameterizations of those modules. At the same time, MPL's software environment must be efficient enough to meet the demands of real-time navigation research.

The need to balance between flexibility and efficiency has led us to design a software environment around the ISR3, an in-memory database that allows users to define structures for storing visual data, such as images, lines and surfaces [Draper 93a,b]. ISR3 serves as a process communication interface, so that, for example, lines produced by one module can be used by another, even if the second module is run later or on a different processor than the first. ISR3 also provides modules with efficient spatial access routines for visual data, and protects data from being simultaneously modified by two or more concurrent processes. A graphical programming interface allows programmers to easily sequence modules and modify their parameters.

Our work on planning has concentrated on plan execution rather than plan generation; thus, we assume that plans are developed by the user (by hand) or by an appropriate planning subsystem. With this goal in mind, a control system, called the Script Monitor, has been implemented to support real-time execution of plans [Rochwerger et. al. 94]. The task of autonomous navigation, as with many other complex tasks, can be decomposed into specific subtasks like road following, obstacle avoidance and landmark recognition. In order to achieve a fully autonomous capability, the solutions to all these subproblems must be integrated into a coherent system. This paper focuses on the preliminary work done on the problem of integration for the UMass Mobile Perception Laboratory (MPL). Since each of the

subproblems is still a field of active research in which approaches and solutions may change rapidly over time, the integrated system should be flexible enough to allow testing different subsystems and different control strategies.

To achieve the desired flexibility we have implemented the control system as a "programmable" finite state machine (FSM), in which the states represent the different modes of operation of the system (behaviors) and the state transitions correspond to the system's reactions to events (either external or internal). The composition of the states and the transitions is not fixed, i.e., the FSM can be tailored to the particular task the system is trying to achieve.

3. Integration via the ISR Database.

MPL's database is built around ISR3, an in-memory database for computer vision. Historically, ISR (an acronym for *intermediate symbolic representation*) has been the name of a series of symbolic databases for vision developed at the University of Massachusetts [Brolio et. al. 89]. One version, ISR1.5, is now commercially available as part of KBVision™ [Williams 90], while the most recent version, ISR3, is used on-board the MPL. The ISR databases reflect a belief that computer vision requires more than image-like arrays of numerical data; computer vision depends on symbolic representations of abstract image events such as regions, lines, and surfaces, and on mechanisms for efficiently accessing data objects by a model-based system under various types of constraints (such as spatial proximity). Although each version of ISR has been a refinement of its predecessor, they all assume that visual procedures operate on symbolic records, called tokens, or on groups of tokens, and that visual procedures manipulate tokens both for internal computations and for exchanging data with other procedures.

ISR3 is an in-memory database for C structures. It reads a C header file of structure definitions when it is first initialized, and records the size of the structures (called *tokens*) and the data types and offsets of their fields. ISR3 then establishes a database for these structures, establishing queues for allocating and freeing tokens, semaphores for preventing simultaneous access, and functions for storing and retrieving tokens. Once one process has initialized ISR3, other processes can attach to it (since all tokens are kept in shared memory), making ISR3 a communication mechanism as well as a data store.

3.1 ALVINN

In order to emphasize ISR3's role as an integration mechanism, its communication, synchronization, and data storage capabilities will be described in the context of specific MPL tasks. One of MPL's most basic tasks is to drive down roads, using the ALVINN neural-net road-following system [Pomerleau 90, 92]. Developed at Carnegie-Mellon University, ALVINN is a neural network with a single hidden layer that produces steering vectors from reduced (32x30) and color-compensated intensity images[1]. When run alone it implements a simple road following behavior by grabbing images from a forward-looking camera and sending commands to the (low-level) vehicle controller to correct for drift and turns in the road in order to keep the vehicle on course.

On the MPL ALVINN is almost never run in isolation, however. At the very least it is run with an obstacle detection program to ensure the safety of the vehicle (otherwise ALVINN is more than happy to run into obstacles). ALVINN is integrated with the obstacle detection program by interposing an arbiter between ALVINN and the vehicle controller. The arbiter takes the steering vectors produced by ALVINN and combines them with the results of obstacle detection to make sure the vehicle avoids obstructions.

ALVINN is integrated with the rest of MPL's software by declaring its steering vectors to be ISR3 tokens stored in shared memory. ALVINN can then be tested in isolation from other systems by having the controller read ALVINN's steering tokens directly, and combined with other systems by having the arbiter retrieve ALVINN's tokens and produce its own (modified) steering tokens for the controller. A similar principle can be applied to image acquisition, with a data server producing image tokens and storing them in shared memory for ALVINN (or any other process) to retrieve.

3.2 Landmark Recognition and Positioning

Although ALVINN demonstrates how ISR3 can help integrate software modules, including modules developed at other sites, it does not exercise all of the ISR3 capabilities. MPL's

[1]Properly speaking, an in-memory database is called a data store.

landmark recognition system is a combination of four software modules that together match 3D landmark (or object) models to images and determine the position and orientation of the camera relative to the landmark. The first module uses color and texture information to limit the search for the landmark to a specific region of interest (ROI), the second extracts straight (2D) line segments from the ROI, the third projects the 3D landmark model according to the estimated viewing point and determines which (3D) lines should be visible, and the fourth matches (2D) image lines to visible (3D) model lines and determines the relative position of the vehicle to the landmarks in the world coordinate system.

As with ALVINN and the arbiter, ISR3 is the communication mechanism that allows the four landmark recognition modules to exchange data, this time by declaring ROIs, (2D) image lines, (3D) model lines and (3D) faces to be ISR3 tokens. The landmark recognition modules are more typical of vision applications than ALVINN, however, in that they produce large numbers of tokens which are typically accessed by name, feature value or spatial location. Spatial access is particularly important for the matching modules, which must repeatedly access image lines near the projections of model lines (according to the current pose estimate).

Most landmark recognition tokens, particularly ROIs and image lines, exist for only a short period of time (in this case the time required to process one image) and should then be deallocated. Consequently, file I/O is not optimized; although data is sometimes saved for later analysis in the lab, most data can be kept in memory and then erased without ever being saved to disk (ignoring the effects of paged virtual memories). Token allocation and deallocation, on the other hand, are critical, which is why ISR3 maintains its own token queues.

The landmark recognition processes are also typical of many vision application programs in that they both produce and consume sets of tokens rather than single tokens. The matching module, for example, finds (potentially many-to-many) correspondences between sets of image lines and sets of model lines. Therefore most storage and retrieval commands in ISR3 are in the form of set operations, such as a request to access "all long, straight lines in the upper corner of an image". Synchronization is provided, not at the level of individual tokens, but of sets of tokens, so that processes may iterate over sets of tokens without having to lock and unlock each token individually. Special facilities for optimizing spatial retrieval over individual or arbitrary sets of tokens are also provided, as are macros for iterating over the tokens in a set, and functions for taking the union, intersection and differences of sets[2].

Although most tokens are temporary, ISR3 also provides permanent storage for those few sets of tokens (critical features, updated maps, etc.) that correspond to significant results and should persist over time. Model acquisition processes, such as one described by Sawhney [Sawhney 93] produce 3D models of the type used for object recognition (i.e. 3D points, lines and faces). These models, once learned, should be permanently stored, and the landmark recognition processes should always have access to the most recent version of any model. With regard to these tokens, therefore, ISR3 serves as a permanent data base system that provides storage for, and structured access to, long-term data.

3.3. ISR3 Protection and Memory Management Mechanisms

Although ISR3 acts as a general database system, it has been optimized for real-time vision research by reducing overhead wherever possible while still supporting those functions most often used in computer vision. For example, one task of a database in a multiprocess environment is to stop processes from accidentally overwriting or destroying each other's data. In a typical computer vision application, hundreds or thousands of tokens may be in memory at one time. If multiple processes have uncontrolled access to these tokens and modify them, unpredictable interactions will cause elusive and non-repeatable bugs. On the other hand, it is not uncommon for a process such as the model matcher to access hundreds of tokens at a time. If it had to lock and unlock a token each time it reads a feature value, protection would become unacceptably expensive, especially given the relatively slow speed of semaphores under UNIX.

A compromise used in ISR3, therefore, was to associate semaphores with sets of tokens. When a

[2]The complement of a set is not well defined, since there is an infinite universe of possible tokens.

set of related tokens is created, for example the set of lines extracted from a ROI, a semaphore is allocated to protect those tokens. Any ISR3 function that accesses that set of tokens will first check the semaphore; ISR3 access functions that create subsets of a defined set of tokens assign the same semaphore to the subset that is used for the parent set. If users access tokens surreptitiously through C pointers[3], they are expected to lock the semaphore before accessing the first token and to unlock it after the last token access. As a result, as long as users do not circumvent its safeguards, ISR3 is able to provide process synchronization with very little overhead.

Similarly, ISR3 provides a level of memory management on top of the UNIX operating system. For non-real-time, file-based systems, memory management is not a critical issue; for continuous, real-time systems, however, it is crucial. ISR3 applications operate in real-time loops, allocating new tokens on each iteration. Memory allocation must be rapid, and must avoid fragmenting memory (as repeated calls to *malloc* would). Memory must be recycled, with space allocated to old tokens being reassigned to new ones once the old data is no longer needed. ISR3 satisfies these requirements by providing token buffers (at a level hidden from the user). Calls to create a token actually allocate one from the buffer for that token type, and freed tokens are returned to the appropriate buffer. For users who store tokens in hierarchies, ISR3 also provides functions for tracing through a hierarchy and freeing all the tokens in it, so that, for example, one can free all the memory associated with an image once the image is no longer current.

4. Executing Reactive Behaviours: Scripts and the Script Monitor

The term behavior has generally been used in the literature to describe processes that connect perception to action, i.e., a behavior senses the environment and takes an appropriate action based on what was perceived. A combination of behaviors is also called a behavior; thus, a complex behavior can be achieved by combining simpler behaviors. In Brooks' subsumption architecture [Brooks 86], the task of robot control is decomposed into levels of competence; each level, in combination with lower levels, defines a behavior. In their Distributed Architecture for Mobile Navigation system (DAMN), Payton, Rosenblatt and Keirsey [Payton 86; Payton et. al. 87] refer to behaviors as very low level decision-making processes which are guided by high level plans and combined through arbitration. In their DEDS work, Ramadge and Wonham [Ramadge 89], as well as Rivlin [Rivlin], events are considered the alphabet, Σ, of a formal language. A behavior is then a sequence of events, or a string over Σ^*. Note that in this terminology every prefix of a string is also a behavior, i.e., the sequential combination of behaviors is a behavior.

These definitions, although consistent, can be confusing - the same term is used for individual processes and for the composition of these processes. We have chosen to think of a behavior as a mode of operation [Payton et. al. 90], in which several perception-action processes [Draper 94] are executed concurrently. Each process converts sensory data into some kind of action (either physical or cognitive), and at any time may generate an event - a signal to let the system know that "something" significant has occurred. All inter-process communication is achieved through a global blackboard -a section of shared memory accessible to all processes. The blackboard is built on top of the ISR3 which, as described earlier, provides a very efficient memory management mechanism (on top of UNIX) and a set of primitives necessary for shared memory based communication.

4.1. A Finite State Machine Representation for Behaviors

An autonomous system must react to events by changing its behavior; hence the sequence of behaviors actually executed depends upon the sequence of events. Since the latter is unpredictable, so is the former. To program such a system, one must specify which processes constitute a behavior and for each possible event describe the system's reaction. In order to specify such a system, a finite state machine (FSM) formalism has been chosen, in which the states represent behaviors and the transitions reactions to events.

As a simple illustrative example, the following set of statements describe a system that will drive on

[3]Since ISR3 stores arbitrary C Structures, there is no way to prevent users from accessing tokens without using ISR3 functions, once the user has obtained one pointer into the database.

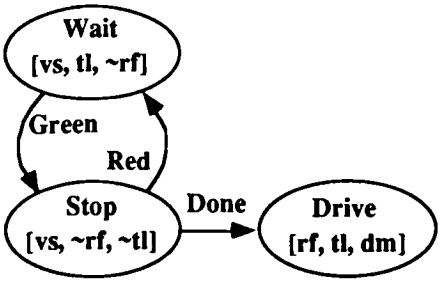
the road while obeying traffic lights, until a given distance is traveled:

While driving down the road
if the traffic light turns red, wait.
if goal is reached, stop.

While waiting at the traffic light,
if it turns green, start driving.

These statements correspond to the simple FSM in Figure 3a; note that a state represents a behavior or mode of operation, i.e, a set of concurrent perception-action processes. This example can be implemented with four perception-action processes: follow the road (*rf*), check for traffic lights (*tl*), monitor the distance traveled (*dm*), and stop the vehicle (*vs*). Listed below the state name are the perception-action processes that should be run and killed (marked with a ~) in that state.

Based on the notion of behaviors represented as states of a finite state machine, a Behavior Description Language (BDL) was designed and implemented. Behaviors are described as two sets of perception-action processes, and a transition table. The run set specifies the minimum set of processes that form the behavior; the kill set specifies those processes that should not be running for the correct execution of the behavior. The choice of two sets implies that processes that were running when the behavior started will continue to run unless explicitly killed. The transition table specifies what to do for each of the valid events (events not specified in the state description are not valid). The representation of our simple example expressed in the BDL is shown in Figure 3b. First, the perception-action processes available are listed. Then the set of states (or behaviors) and the set of events are declared. Finally, a description of each state is provided; parameters required for the perception-action processes associated with each state are fetched from the blackboard prior to their initiation. For a more complete example, see Section 4.4.



(a)

```
PROCS = {
    rf      "DriveOnRoad"
    tl      "CheckTrafficLight"
    vs      "VehicleStop"
    dm      "DistanceMonitor"}

STATES = {drive, wait, stop}

EVENTS = {red, green, done}

WHILE drive(d) {
    SET distance = d;
    RUN rf, tl, dm;
    EVENT red GOTO wait;
    EVENT done GOTO stop;}

WHILE wait () {
    KILL    rf;
    RUN     vs, tl;
    EVENT green GOTO drive;
```

(b)

Figure 3. Script of a simple driving system. (a) Finite state machine (FSM) representation. (b) Behaviour Description Language (BDL) representation.

4.2. Scripts - Augmented finite state machines

The simple FSM model discussed in the previous section has been augmented in two ways. First, a *fetch-goal* state was added to the set of states as a mechanism for compacting the representation. The system starts in the *fetch-goal* state where it reads goals (in terms of behaviours) from a precompiled plan. When a goal is retrieved, the script monitor writes relevant blackboard messages into the blackboard before creating the perception-action processes associated with the state. Once a perception-process is running, it looks for its parameters in the blackboard, which is implemented within the ISR structure described earlier. The *fetch-goal* state differs from all other states in two ways: (1) transitions out of the state are unlabelled since the next state is explicitly specified in the goal retrieved from the script; (2) Unless the plan is empty, the kill and run sets are ignored.

With these change, a script S is formally defined as the eight-tuple $(P,Q,E,M,\delta,\kappa,\rho,G)$, where:

- P is the set of available perception-action processes.
- Q is the set of states ($Q = B \cup$ *fetch-goal*, where B is the set of behaviors).
- E is the set of possible discrete events (transitions in the FSM).
- M is the set of valid blackboard messages.
- δ is the transition table, $\delta : B \times E \rightarrow Q$.
- κ is the kill table, $\kappa : B \times P \rightarrow \{0, 1\}$.
- ρ is the run table, $\rho : B \times P \rightarrow \{0, 1\}$.
- G is the plan expressed in terms of subgoals. Each subgoal is of the form $<b_g,M_g>$, for $b_g \in B$ and $M_g \subseteq M$.

Scripts can be generated by an automated planner, or by hand (using BDL).

4.3. The Script Monitor

The script monitor is in charge of "high level" control[4]: reading, interpreting, and executing BDL scripts. Essentially, the script monitor is a *plan execution system* [Georgeff 90] similar to PRS [Ingrand et. al. 92; Lee et. al. 93] in some aspects. The monitor does not perform any direct action on the vehicle controller by itself; rather, it controls the set of running processes which do take direct action.

The script monitor consists of two modules, an *interpreter* and an *execution system*. The interpreter takes a BDL script S and builds the transition (δ), kill (κ) and run (ρ) tables. After this, the subgoals in S's plan are stacked into the *execution stack* G. The execution system simulates a finite state machine as shown in Table 1.

Clearly, for the system to complete the task in G, the following must hold:

$$\forall b \in Q \; \exists s_b \in E^+ \text{ s.t. } \delta'(b,s_b) = \textit{fetch-goal}$$

where δ' is the transition function applied to a sequence of events [Hopcroft and Ullman 79].

[4] In this context, ``high level'' control is used to differentiate the control of processes from the "low level" control of the vehicle actuators.

4.4. A More Complete Example and an Experiment

A set of perception-action processes have been implemented for the MPL. These include:

- Vehicle pose determination based on landmark model matching [Beveridge 93; Draper 93b; Kumar 92,93].
- Neural-network road following (ALVINN) [Pomerleau 90].
- Servo-based steering [Fennema and Hanson 90; Fennema 91].
- Obstacle detection via stereo [Badal et. al. 94].
- Reflexive obstacle avoidance [Ravela et. al. 94].
- A distance monitor.
- Turning via dead reckoning.
- Servo to compass heading.
- Harmonic function path planner [Connolly et. al. 92,93]

In the future, additional processes will be added, including landmark tracking, recognition and modeling of natural landmarks, automatic landmark extension, etc.

Using a subset of these processes, an experiment was designed in order to demonstrate the capabilities of the vehicle and the performance of the independent perception-action processes. The following script was successfully tested on the vehicle at the UMass test site:

(1) Drive on the road, while avoiding obstacles, for x meters.
(2) Estimate vehicle position using landmarks.
(3) Drive on the road, while avoiding obstacles, for y meters.
(4) Estimate vehicle position using landmarks.
(5) Turn left (at the experimental site this command is a transition to off-road navigation).
(6) Drive off road (by servoing on a compass heading), while avoiding obstacles, for z meters.

1. $b_g \rightarrow$ *fetch-goal* (Start at the fetching state)	
2. if (b_g = *fetch-goal*) then	
(a) if G is empty then $\forall p \in P$	
i. kill (p)	(Terminate ALL processes)
ii. if ($\rho(b_g, p) = 1$) then run(p)	(Run cleanup processes)
iii. stop	(S was successfully executed)
(b) $<b_g, M_g> \leftarrow$ pop(G)	(Fetch next goal)
(c) blackboard $\leftarrow M_g$	(Write blackboard messages)
3. $\forall p \in P$ if $\kappa(b_g, p) = 1$) then kill(p)	Terminate processes)
4. $\forall p \in P$ if ($\rho(b_g, p) = 1$) then run(p)	(Create processes)
5. Wait for an event $e \in E$	(Wait)
6. $b_g \leftarrow \delta(b_g, e)$	(React to event - follow transition table)
7. goto 2	(Repeat until all goals are achieved)

Table 1. Script Monitor Execution Subsystem

The perception-action processes involved in this example include pose estimation (*pe*), road following (*rf*), obstacle detection (*od*), obstacle avoidance (*oa*), servoing to a compass heading (*se*), distance monitor (*dm*) and dead reckoning turning (*dt*). The full BDL script for this coordinated action, with x = 100, y = 150 and z = 50, is:

```
PROCS = {
    pe      "PoseEstimate"
    rf      "RoadFollow"
    od      "ObstacleDetect"
    oa      "Obstacle Avoid"
    se      "Servo"
    dm      "DistanceMonitor"
    dt      "DeadReckoningTurn"
    vs      "VehicleStop"}

STATES={drive-onroad, drive-offroad, turn,
        compute-pose, avoid-obstacles}

EVENTS = {success, obstacles, clear}

WHILE drive-onroad (dist) {
    SET distance = dist;
    RUN rf, od, dm;
    EVENT success GOTO compute-pose;
    EVENT obstacle GOTO avoid-obstacles;}

WHILE drive-offroad (dist) {
    SET distance = dist;
    RUN se, od, dm;
    EVENT success GOTO compute-pose;
    EVENT obstacle GOTO avoid-obstacles;}

WHILE turn (dir, dist) {
    SET direction = dir;
    SET distance = dist;
    RUN dt, dm;
    EVENT success GOTO fetch;}

WHILE avoid-obstacles ( ) {
    KILL rf, se;
    RUN oa, od;
    EVENT clear GOTO BACK;}

WHILE compute-pose ( ) {
    KILL rf, se;
    RUN pe;
    EVENT success GOTO fetch;}

GOALS {
    drive-onroad (100);
    drive-onroad (150);
    turn (left, 10);
    drive-offroad (50);}
```

The augmented finite state machine for this script is shown in Figure 4. Note that the figure shows the addition of the *fetch-goal* state discussed earlier. The FSM also shows a potential link to the high level planning system (not yet implemented) which is activated (in this example) by failure of the pose estimation state to localize the vehicle. In this case, the vehicle is considered to be lost - it then stops and initiates planning to resolve its location. Since planning may result in modifications of the goal stack, the addition of the

planning state[5] may theoretically change the model to a push-down automata (PDA).

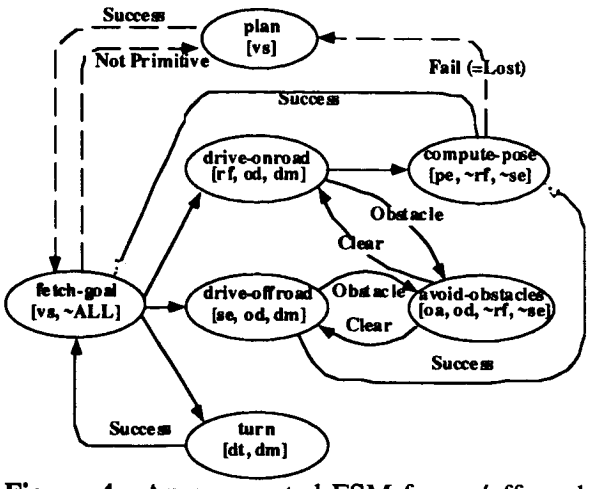


Figure 4. An augmented FSM for on/off road navigation.

5. Summary and Conclusions

One of the requirements for MPL's software environment was that it should support the integration of real-time visual procedures with as little overhead as possible. This meant that the focus of the system design had to be on the two critical areas of data storage/exchange and process control. Data storage and exchange is supported by the ISR3 real-time database/datastore system, which provides a central data repository and communication mechanism for all the perception-action processes running on the MPL. Behaviors are collections of concurrent perception-action processes whose interaction and execution are controlled through a script; MPL's script monitor is a low-overhead control system that switches from one behavior to the next in response to external conditions.

Although limited, the script monitor system in its current form has given us an idea of the complexity involved in building intelligent controllers, particularly in a real-time application domain where safety must be ensured. Encoding system reactions as a finite state machine seems a reasonable approach, but it is not clear how to optimally construct the individual states, i.e. which perception-action processes constitute each state, and how these processes interact with one another.

[5]Or any state that can write to the goal stack.

In the current implementation, all communication between processes is done through the blackboard. In this particular domain, where potentially large amounts of data are shared (images, maps, etc.), the shared memory paradigm seems the most efficient method of communication for processes running on the same machine. But if perception-action processes were to run in a distributed architecture, other means of communication will be necessary (UNIX sockets or a system such as TCX [Fedor 93]). In the current implementation, it was assumed that scarcity of resources was not an issue. However, independently executing concurrent processes which access sensors and send commands to actuators, or otherwise affect other scarce system resources, will inevitably cause problems if resource allocation is not handled correctly.

Resource scheduling and sharing, inter-process communication and real-time control are difficult problems, particularly in a real-time dynamic environment, and efficient solutions to them are essential if robust autonomous systems are to be constructed.

Acknowledgments

This work has been supported in part by Advanced Defense Research Projects Agency (via TACOM), under contract number DAAE07-91-C-RO35, and by the National Science Foundation under grant number CDA-8922572.

References

Badal, S. and B. Draper, "Stereo Obstacle Avoidance on the MPL", forthcoming Computer Science Technical Report, 1994.

Beveridge, J.R., "Local Search Algorithms for Geometric Object Recognition: Optimal Correspondence and Pose", Ph.D. Dissertation, University of Massachusetts at Amherst, Computer Science Department Technical Report TR93-71, 1993.

Brolio, J., B. Draper, R. Beveridge, and A. Hanson, "The ISR: An Intermediate System Representation for Computer Vision:, IEEE Computer, 22(12), 1989, pp. 22-30.

Brooks, R. A., "A Layered Robust Control Systems for a Mobile Robot," 2(1), 1986, pp. 14-25.

Connolly, C., and Grupen, R., "Harmonic Control", Proc. of the International Symposium on

Intelligent Control, Glasgow, Scotland, August, 1992, pp. 503-506.

Connolly, C., and Grupen, R., "On the Applications of Harmonic Functions to Robotics", Journal of Robotic Systems, Vol.10, No.7, October, 1993, pp. 931-946.

Draper, B., Hanson, A., and Riseman, E., "ISR3: A Token Database for Integration of Visual Modules", Proc. ARPA Image Understanding Workshop, Washington, D.C., April 1993a, pp. 1155-1161.

Draper, B., S. Buluswar, A. Hanson, E. Riseman, "Information Acquisition and Fusion in the Mobile Perception Laboratory," Proc. of Sensor Fusion VI, Boston, MA, Sept. 1993b, pp. 175-187.

Draper, B., A. Hanson, and E. Riseman, "Integrating Visual Procedures for Mobile Perception," in Experimental Environments (H. Christensen, Ed.), World Scientific Press, to appear; also to appear in CGVIP:IU, March 1994.

Fedor, C., "TCX Task Communications (Version 7.7)", Robotics Institute, Carnegie Mellon University, January 1993.

Fennema, C., "Interweaving Reason, Action, and Perception," Ph.D. Dissertation, University of Massachusetts at Amherst, Computer Science Department Technical Report TR91-56, 1991.

Fennema, C. and A. Hanson, "Experiments in Autonomous Navigation", Proc. 10th International Conference on Pattern Recognition, 1990, pp. 24-31.

Georgeff, M. P. Planning. In Readings In Planning, Morgan Kaufmann, 1990, pp. 5-25.

Hanson, A., and Riseman, E., and Weems, C., "Progress in Computer Vision at the University of Massachusetts", Proc. ARPA Image Understanding Workshop, Washington, D.C., April 1993, pp. 39-47.

Hopcroft, J. E. and J. U. Ullman. Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

Ingrand, F. F., M. P. Georgeff, and A. S. Rao, "An Architecture for Real-Time Reasoning and System Control, IEEE Expert, 1992, pp.

Kumar, R., and Hanson, A., "Robust Methods for Estimating Pose and a Sensitivity Analysis", CGVIP:IU to appear.

Kumar, R., "Model Dependent Inference of 3D Information from a Sequence of 2D Images", Ph.D. Dissertation, University of Massachusetts at Amherst, Computer Science Technical Report 92-04, February 1992.

Lee, J., M. J. Huber, E. H. Durfee, and P. G. Kenny, "UM-PRS: An Implementation of the Procedural Reasoning Systems," Artificial Intelligence Laboratory, Univ. of Michigan, 1993.

Payton, D.W., "An Architecture for Reflexive Autonomous Vehicle Control", Proc. IEEE Robotics and Automation Conference, 1986, pp. 1838-1845.

Payton, D. W., K. Rosenblatt, and D. M. Keirsey, "Plan Guided Reaction", IEEE Trans. on Systems, Man, and Cybernetics, 1990, pp. 1370-1382.

Pomerleau, D. A., "Neural Network-Based Autonomous Navigation," in Vision and Navigation: The CMU NavLab (C. Thorpe, ed.), Kluwer Academic Publishers, 1990.

Ramadge, P. J. and W. M. Wonham, "The Control of Discrete Event Systems", Proc. of the IEEE, 77(1), January 1989, pp. 81-98.

Ravela, S., and Draper, B., "Reflexive Obstacle Avoidance on the MPL", forthcoming Computer Science Technical Report.

Rochwerger, B., C. Fennema, B. Draper, A. Hanson, and E. Riseman, "Executing Reactive Behavior for Autonomous Navigation," forthcoming technical report, Computer Science Department, University of Massachusetts at Amherst, 1994.

Rivlin, E., "The DEDS Formalism for Systems with Vision," University of Maryland.

Sawhney, H., "Spatial and Temporal Grouping in the Interpretation of Image Motion", Ph.D. Dissertation, University of Massachusetts at Amherst, Computer Science Technical Report 92-05, February 1992.

Weems, C. Herbordt, M., Dutta, R., Daumueller, K., Weaver, G., and Dropsho, S., "Status and Current Research in the Image Understanding Architecture Program", Proc. ARPA Image Understanding Workshop, Washington, D.C., April 1993, pp. 1133-1140.

Williams, T., "Image Understanding Tools," 10th International Conference on Pattern Recognition, Atlantic City, NJ, June 1990, pp. 606-610.

AIAA-94-1243-CP

THE 4-D APPROACH TO VISUAL CONTROL OF AUTONOMOUS SYSTEMS

Ernst D. Dickmanns
Universitaet der Bundeswehr München
D-85577 Neubiberg, Germany

Abstract

Based on experience with real-time image sequence processing systems in the application areas of vehicle docking, road vehicle guidance, AGV's on the factory floor, aircraft landing approaches and of dynamical grasping of free floating objects in space with remote control from the ground including long delay times, an efficient, distributed, expectation- as well as object-based general dynamic vision system architecture has been developed. Parallelization is structured according to physical objects, the characteristics of which with respect to 3-D shape, motion behavior, visual appearance and all other significant properties in the task context are represented internally in generic form. Both differential representations for state estimation as well as behavior control, and integral ones for mission planning, mission control and monitoring are being used. References to detailed reports on all application areas mentioned are given.

Introduction

The sense of vision is ***the*** predominant source of information for intelligent motion control in biological systems; why has it been missing in technical systems almost entirely until very recently? There are at least two basic reasons: First, human visual capabilities are well developed, and similar real-time performance on the technical side requires computing capabilities not nearly available until about a decade ago; the data flow in a color video signal is of the order of magnitude 10^7 Bytes per second (10 MB/s) while clock rates of computers are between 10 and 100 MHz. Assuming 10 to 100 operations per data point (or 'picture element') in the image (this will be abbreviated in the sequel as 'pel' or 'pixel') it is immediately seen that many parallel processors are needed for real-time performance just for the image sequence processing part, let alone dynamic scene understanding, control computation and mission monitoring.

The second reason is, that -unlike in biological systems- digital image processing started from static single image evaluations as in remote sensing applications. Until the early 80ies, when researchers from the field of control engineering moved into this newly developing field of image sequence evaluation, the approach to this field has been dominated by 'quasi-static' thinking; time has been introduced through the backdoor by differencing between images and starting from so-called 'optical flow' information.

However, from linear systems theory in the field of trajectory reconstruction based on noise corrupted measurements, recursive estimation techniques have been known since the early 60ies[1] which allow to substitute knowledge about the real-world processes to be observed for missing or poor-quality data. These so-called 'dynamical models' represent temporal dependencies explicitly as side constraints for data interpretation during process evolution over time. Exploiting these constraints systematically in conjunction with spatial shape characteristics resulted in increased image sequence processing efficiency by orders of magnitude[2].

This is due to the fact that temporal predictions using the dynamical models allow to control both assignments of image regions to parallel processors and the extractions of features by special algorithms in limited search regions depending on the situation encountered; all of this is geared to objects of specific classes for which corresponding generic knowledge is represented in 'object processor groups'. This leads to efficient local communication structures and to a modular system design[3].

At UniBwM this approach has been applied to half a dozen different guidance and control tasks. It became well known in the field of visual guidance for autonomous land vehicles where surprising performance levels have been achieved with very moderate computing power. Over the last several years the approach has enjoyed increasingly widespread use worldwide; there are many variants documented in the literature and their number is increasing rapidly [4-13]

In this paper, a survey is given on the general method as it presently stands at UniBwM. The following section covers the method proper featuring the basic ideas like 4-D representation, orientation towards physical objects, expectations and prediction error feedback as well as the central role the Jacobian matrices play for the relationship between image features and object state components; with increasing numbers of sensors and of objects in the scene analysed, the management scheme of the perception system had to

Copyright ©1993 by EDD. Published by the AIAA, Inc. with permission

become capable of dealing with occlusions, partial observability due to aspect conditions and with model switching on top of the basic distinction between initialisation and tracking phases.

The applications to mobile robots discussed in a survey fashion, following this display of the method, are: rendezvous and docking (planar reaction controlled satellite docking, and spatial autonomous landing approaches of aircraft), surface vehicle guidance with local reactions (road runner including obstacle avoidance or 'intelligent cruise control') and global mission control (landmark navigation both on the factory floor and on an outside road network).

The dynamical grasping experiment during Spacelab mission D2 in May 1993 concludes this application survey; with sensors and the robot arm as effector on board the Space Shuttle Columbia, and with the computers on the ground this teleoperated ('remotely brained') system has achieved the first capture of a free-floating object in space by delayed visual feedback (5.8 seconds !).

The 4-D approach

The 4D approach to dynamic machine vision exploits dynamical models in the form of state transition and control effect matrices for sampled data systems with cycle times as multiples of the basic video cycle time (20 ms in Europe, 16 2/3 ms in the USA); the recursive state estimation methods well known in systems dynamics for smoothing both process and measurement noise are combined here with 3-D shape representations of objects through well visible edge features and with perspective projection of these spatial edge elements into the image plane. The corresponding Jacobian (sensitivity) matrix of feature positions in the image plane with respect to changes in object-state components in 3-D space is used for bypassing the ill-posed nonlinear problem of direct perspective inversion. Instead, the least squares Kalman filter algorithm for noise reduction indirectly also performs this inversion in a sufficiently accurate approximate manner, recovering the third dimension (depth) lost during perspective projection, by temporal continuity conditions in conjunction with the dynamical model used for prediction.

Due to the relatively high temporal frequency of 12.5 to 25 Hz for image sequence interpretation, the underlying linearizations of all nonlinear relationships are sufficiently good; only the last image of the sequence needs be worked with, thereby avoiding storage and retrieval problems with previous ones. This enormously alleviates the data handling problem; no optical flow needs be computed. However, the *spatial* velocity components are obtained by smoothing numerical operations.

This prediction error feedback scheme for each object leads to a servo-maintained internal representation duplicating all essential aspects of the real-world subprocesses being individually observed and analysed. The integral interpretation in 3-D space and time has led to the name '4-D approach'. Figure 1 shows the resulting block diagram for a single imaging sensor and multiple objects besides the own vehicle to be controlled.

In parallel to the real world (shown in the upper left block) with motion processes happening in 3-D space and time, an internal representation, also in 3-D space and time but limited to the most essential aspects for the actual task at hand, is built up in the interpretation process by prediction error feedback ('internally represented world' in upper right block of fig.1). There is a fundamental difference between the initialisation phase when a new object is being discovered, and the tracking phase when temporal continuity conditions yield a good guideline for understanding the evolution of the dynamical scene observed. The basic ideas will be discussed in turn.

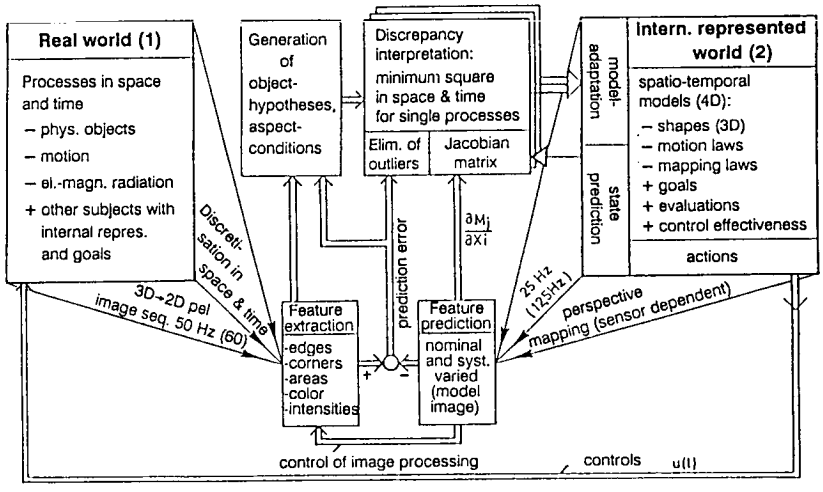


Fig. 1: Basic prediction error feedback scheme for single object and multiple imaging sensor

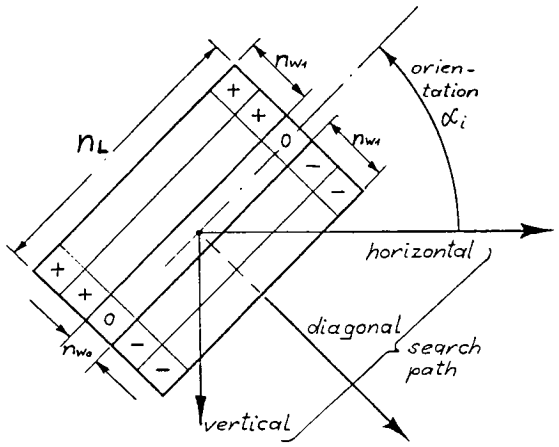
Basic ideas

Five essential elements constitute the base of the approach during continuous observation of a moving object (tracking phase):

Edge element features: The most basic element to efficient image sequence processing in a steadily changing environment is to fully exploit the difference between data and information. A uniformly grey image contains the same amount of data as a page with text and pictures from complex scenes; however, in the former case a human observer completely describes the information content by two words: 'uniformly grey', while for the latter one he may have to talk for several minutes in order to convey at least the most essential aspects. If the image would contain two differently colored areas with a curved boundary, the most economical way of capturing the information content in a symbolic description would be to formulate the boundary between the colored regions by the geometry of a line (straight segments or curves with given curvature along the arc length) and by specifying the homogeneous areas by two color symbols; again, the information exhaustively describing the image can be coded in orders of magnitude less data as compared to the pixels involved. Using tangent direction information and points of discontinuity (corners) seems to be an efficient coding scheme for boundaries in an image. Supposedly, this is the reason behind nature having developed the capability of doing just this in some of the high performance biological vision systems (striate cortex in V1).

Looking for dark-to-bright transitions (or vice-versa) irrespective of the absolute brightness level or spectral information content makes these systems less dependent on threshold values and thus more robust. Confining these tangent direction measurements to closely spaced discrete points in the image plane yields a natural discretisation allowing to construct both smooth curves from assumed linear changes of curvature along arc length in a differential geometry interpretation (corresponding to third order polynomials in Cartesian space over not too large arc lengths) and sharp corners when tangent directions are far apart even though their centers are closely spaced[14]. By these tangents (edge elements) any shape can be represented by corresponding feature groupings; in a multiple scale concept, mask operators for feature extraction can detect dark-to-bright transitions on several scales thereby allowing object detection and characterization with different resolutions; for example, for many practical purposes in vehicle guidance it is sufficient to characterize obstacles by the encasing rectangle or box. In bifocal vision with different focal lengths evaluated simultaneously, this gives an easy choice for either fast tracking or more precise shape determination.

Therefore, the family of ternary edge feature extractors as shown in fig. 2 is used as the predominant image processing tool in the context of the 4-D interpretation scheme; it has been developed over a decade of work in real-time dynamic scene understanding[15-17]. The mask parameters are dynamically controlled by the object recognition process taking the 4-D representation and perspective mapping into account, thereby realizing a fast feedback loop from high-level interpretation to low-level feature extraction; it is especially this feature which makes the tracking phase so efficient.



$\alpha_i = i\frac{\pi}{n_{L-1}}$; $i = 0, \ldots, n_{L-2}$; $n_L = 3, 5, 9, 17$

Fig. 2: Operator family for edge feature extraction

Representation in 3-D space and time directly: No basic representations are performed in the image plane; at the earliest time possible it is tried to jump from a feature distribution supposed to belong to a single object to an object hypothesis in 3-D space and time. Since perspective projection is the link between spatial shape, relative orientation as well as position, and the shape in the 2-D image plane, always both object shape and aspect conditions have to be hypothesized in conjunction. Shape invariants of moving rigid bodies are in 3-D space and not in the image plane; simple motion descriptions also are more easily encountered in 3-D space than in the image plane where both motion and shape *together* yield relative image feature distribution from frame to frame. In addition, motion behavior in 3-D space may be as characteristic for an object as its shape; this leads to the third essential element:

Orientation towards physical objects: Knowledge about the real world is attached to objects which serve for structuring complex scenes. Similar properties or shapes lead to the definition of object classes characterized by generic forms and functions; other attributes may be appended depending on the task at hand (e.g. color, texture). For subjects, defined as objects with the capability of self-initiated locomotion[18], stereotypical motion characteristics may give independent cues to recognition besides static shape. In general, the centroid of features from an object yields information for translational motion, while rotational motion and shape may be derived from systematic changes

of feature positions around the centroid, that is from differences between feature positions in the image.

Once an object and its motion has been recognized, the continued observation can be made much more efficient by the fourth basic element:

Expectations and prediction error feedback: The dynamical model of a process (e.g. a moving rigid body) may be given to first order by the vector difference equation

$$x[(k+1)T] = A(k)\,x[kT] + B(k)u\,[kT] + v[kT], \qquad (1)$$

where x is the state vector of dimension n, k is the time index for the actual state, T is the cycle time, A is the state transition matrix ($n \cdot n$), B the control effect matrix ($n \cdot r$), u the r-vector of control variables, and v represents process noise with covariance matrix Q. Predictions, of course, are made disregarding the noise term. After prediction the time index k is increased by 1 and from the predicted state x^* in combination with the shape description the features to be measured in the next image are obtained from applying the forward perspective projection equations; for this purpose, first, the spatial positions and orientations of edge elements have to be computed by combining position and angular orientation of the body-fixed coordinate system having its origin at the object center with the shape description in these coordinates.

Then, the perspective mapping equations containing all translations and rotations between the body-fixed and the camera coordinate system (in x^*) as well as the camera parameters p have to be applied to all well visible edge features in order to obtain the predicted (horizontal and vertical) feature positions in the image:

$$y^* = h(x^*,p)\,, \qquad (2)$$

where dim. (y) depends on kind (edge or corner) and number of features. It is assumed that the error between predicted (y^*) and actually measured feature positions (y) is so small that a linear approximation to eq.(2) captures the essential part of the dependencies between y and x:

$$del_y = y - y^* = dh(x^*,p)/dx^* \cdot (x - x^*) = C \cdot del_x, \qquad (3)$$

where C is the Jacobian matrix of all first order partial derivatives linking state component changes to feature shifts in the image plane. Because of the richness in information contained in this matrix and the central role it plays in the 4-D approach, it will be discussed below as the fifth basic element.

The actual measurement data y from feature extraction will be corrupted by measurement noise w (both from the video signal and from image processing); this noise is assumed to be unbiased and white with covariance matrix R so that the measurement model may be written

$$y = h(x,p) + w. \qquad (4)$$

In order to adjust the internal 4-D representation to the process being observed in the real world, prediction error feedback is used according to the recursive estimation techniques[19] derived from the Kalman filter[1] and its extensions[21]. The new best estimate for the relative object state $\hat{x}$ is obtained by adding to each predicted state component weighted elements depending on the measured prediction error; the weights are determined by the so-called Kalman gain matrix K (or its equivalents in the sequential scheme discussed below) taking the noise characteristics Q and R (confidence in both the underlying process model and the measurements) into account:

$$\hat{x} = x^* + K \cdot (y - y^*). \qquad (5)$$

Note that no special provision is made for perspective inversion; the least squares core of the algorithmic procedure for computing the elements of K takes care of perspective inversion hidden in the prediction step and the Jacobian matrix C. Gain computation is not detailed here for brevity; because of occlusions, varying aspect conditions and perturbations in the imaging process, the length of the measurement vector will change steadily. In order to accomodate this easily, the measurement update is made sequentially for each component; this also saves computing time and has been a standard feature of the 4-D approach from the beginning[21].

From this possibility it can be seen immediately that an update of all state components can be made from just one single measurement input; this may look like magic for people grounded in direct perspective inversion. Though this capability is true - substituting knowledge about the real process for missing data -, too few measurements over an extended period of time will lead to drifts in some state components poorly observable from this measurement, or due to model errors as compared to the actual process observed. However, in spite of this fact the value of this property based on the 4-D model can hardly be overestimated for bridging short periods with insufficient measurements for what cause soever. Even periods without any measurements may be bridged by pure predictions (vanishing second term on right hand side of eq. (5)).

An important point resulting from prediction is the capability to efficiently direct image processing by confining attention to smaller subareas of the image, and by providing information on which algorithms may be most economical in the next image (e.g. mask orientation for edge element extraction).

Central role of the Jacobian matrix: Wünsche[21] developed methods for exploiting the entries into the Jacobian

matrix for other purposes beyond simple recursive state estimation. When computing power is limited, there usually are many more features available for extraction and state estimation than can be handled by the system available or from a price / performance point of view; in this situation it is very essential to be able to automatically select the most rewarding set of features yielding best estimation results in limited time. The entries into the Jacobian matrix, balanced for poorly compatible units for the state variables (like positions in meters and angles in degrees), allow to concentrate computing power for image evaluation in those areas of the image and onto those features by which best accuracy is achieved.

Small values of elements in the balanced Jacobian indicate that the corresponding state component hardly effects the corresponding feature position; if an entire column has small values this means that the corresponding state component hardly affects any measurement quantity; therefore, it can not be expected that this state component can be recovered accurately from the values measured. Likewise, if an entire row has small entries this feature is almost constant and independent of state changes; this feature is not well suited for updating the state vector and may well be eliminated from further processing.

For example, for a rectangular box looked at along a center line almost parallel to four edges (so that the image is a rectangle) it is not possible to recover the exact viewing angle because of the cosine-effect involved. If the size of the box (width, height and length) has to be determined in addition to the relative state, the dimension in viewing direction, of course, cannot be recovered; if the box is turned by 90° this size component is well determinable now while another one, now pointing in viewing direction, can no more be iterated. In the general case, the actual aspect conditions determine which state components or shape parameters can be observed and which ones have to be frozen until the corresponding entries in the Jacobian matrix become large enough again. This determines the perception strategy and -management.

In summary it can be stated that the efficiency in image sequence understanding by the 4-D approach is due to the frequent bottom-up and top-down traversal of the representation hierarchy; this is done each cycle of rather short duration so that the correspondence problem is not too hard. The linear differential models allow to tap well proven system theory. However, this only works for the continuous tracking phase.

Initialisation versus tracking

It is almost impossible to say something meaningful in general to the initialisation problem since it depends very much on the task domain and on the knowledge available to the system. For this reason, the tracking phase for specific task domains has been developed first; as expected, once this capability has been available to a certain extend, it turned out to be relatively easy to jump from feature aggregations discovered in an initial search phase (with very low cycle times) to an object hypothesis for which the tracking capabilities are given[22]. If stable tracking can be established and the prediction errors converge below certain thresholds it is claimed that a motion process involving an object of the class instantiated has been discovered; if this is not the case the hypothesis is rejected and a new one has to be tried until the set available is exhausted. Surprisingly many cases can be handled successfully by this procedure[23,24]; however, many unresolved problems remain, especially when occlusions are involved.

Again, for certain task domains like vehicle recognition on highways, many of these problem situations have been resolved by creating the capability to recognize partially occluded objects, even those appearing from full occlusion like in lane changes of one of two vehicles in front[25]; since only partial information is accessible in these cases, model based recognition involving knowledge about normal sizes of objects, about part hierarchies and about likely motion states is essential.

One often hears the call for more systematic bottom-up hypothesis generation in the initialisation phase; this, of course, would be nice to have. However, considering the discrepancy in computing power required for this type of initialisation in a somewhat complex realistic scene, and the one needed lateron for the tracking phase it is conjectured that - even in the long run - the approach of jumping to (maybe several parallel) hypotheses relatively early and then do a critical evaluation over time exploiting 4-D models may be a sensible way to go. More experience in several task domains is necessary in order to answer this question in a solid way.

The 4-D solution for complex tasks

The basic principles discussed above for the single sensor, case carry over to multiple objects and multi-sensor systems[23,24]. A general scheme for these types of complex real-time systems is given in fig.3.

As in the single object, single sensor case there is just one unifying mental representation for recognizing the situation and for controlling action; however, for each object observed there is a specific process (presently implemented on a dedicated group of processors) with access to a corresponding knowledge base for this object class. In this knowledge base generic background knowledge is stored; besides physical properties with respect to motion (the elements of the dynamical model) specific properties with respect to the different measurement processes are stored.

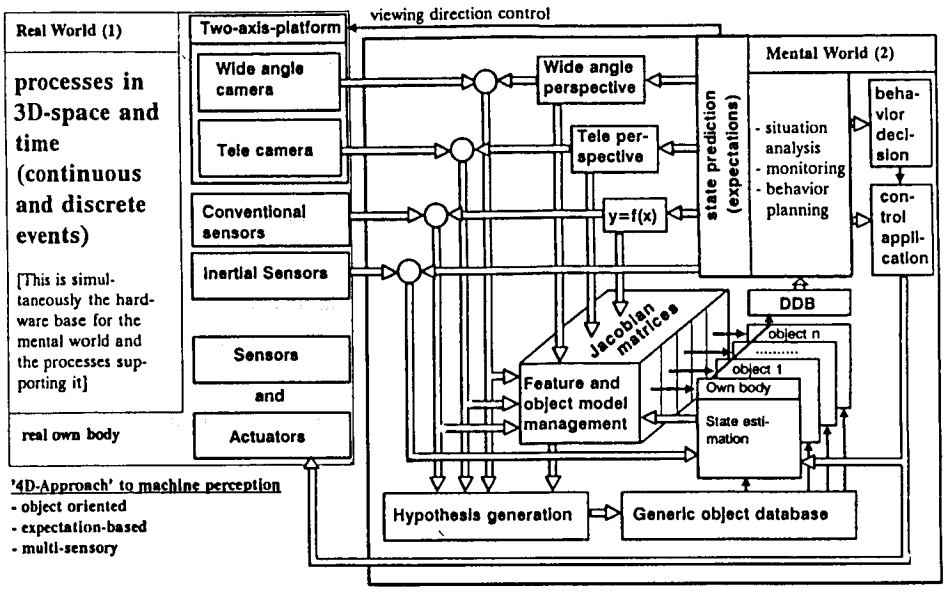


Fig. 3: Multi-sensor, multi object recognition scheme by prediction error feedback

The figure shows provision for four types of sensors, two imaging ones on a pan and tilt viewing direction platform, and two other sets, one for inertial data about the egomotion (lowest group in center) and one for other conventional sensors like odometer, tachometer, steering angle, throttle setting, brake pressure and range sensors or the like.

The inertial sensors may directly feed dynamical models for separating egomotion from visually observed relative motion with respect to another object. Otherwise, vision does allow this separation only when besides this object also a third, static one can be observed simultaneously; accuracy and robustness may be much poorer without inertial data.

For measurement signal interpretation, a Jacobian matrix has to be computed for each pair of object and sensor; in figure 3 this is indicated by the vertical arrows to the diagonal block in the lower right center.

Contrary to the conventional sensors, image sequence evaluation may yield measurement data on several objects in parallel; in fact, one of the difficulties in machine vision is the assignment problem of features extracted to objects in the scene. Grouping image sequence processing according to objects observed, as in the 4-D approach, therefore, requires a perception management subsystem shown also in the lower right center. This is an area of actual research and development; good solutions to this problem will be crucial for high performance machine perception systems.

In order to further decouple real-time fast and safe control from slower activities for situation recognition, two different time scales for image sequence interpretation have been introduced in our system. Besides the fast tracks for estimation of relative position, one each for each object of relevance, based on rather few features per object and working at 25 Hz in the new TIP-system, there is one subsystem, attention controlled from the higher levels, which runs at about 5 Hz and is capable of recognizing objects on a more detailed level; specialists for recognition of typical road vehicles[25] (trucks, vans, passenger cars) and of moving humans[26] (walking, running, bicycling and arm waving) are under development.

Perception management

Vision and inertial measurements do have nice complementary properties: Vision incurs long delay times between data collection and object state estimation in general (100 to 300 ms typically, both in biological and in technical systems). In addition, because of the signal integration in the basic sensing element, motion blur will occur in the image during faster rotation, yielding rate signals derived from image sequences unreliable; on the other hand, inexpensive inertial sensors may yield rather accurate rate signals with almost no time delay. These inertial sensors become expensive when provision has to be made for low drift rates (long term stability). Combining inertial sensors with vision allows to build a flexible system with good overall properties: viewing direction may be stabilized by controlling the platform with the negative angular rate from an inertial sensor, thereby improving the conditions for image evaluation. Visual fixation of the viewing direction onto a well visible set of stationary features allows to solve for the inertial drift problem.

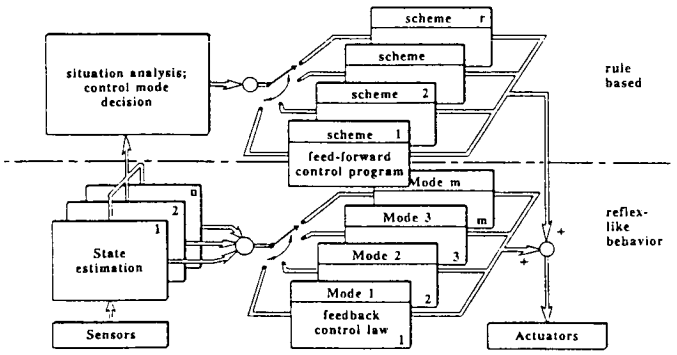
In bifocal vision, two cameras are mounted fixed relative to each other on a gaze control platform; with one wide angle lens for a large viewing angle covered and one tele-lens for high resolution capability within a subarea of the wide angle image, there is a need for viewing direction control in a saccadic mode in addition to the smooth pursuit mode for object tracking: If the tele camera tracks an object, and in the wide angle image a new object of higher interest is discovered, viewing direction should be changed as fast as possible in order to center the new object in the tele-image. With the viewing direction control platform developed[27] a 20° saccade can be performed in about 150 ms; during this fast turn of four evaluation cycles (4·40 ms = 160 ms) no useful information can be derived from the blurred images. Therefore, within these periods the internal representations have to be updated according to the 4-D model exclusively; only after slowdown below a certain angular rate in the vicinity of the coordinates aimed at, image feature extraction will start again with new predicted positions and search ranges. Via a status bit this information is broadcast to all 'object processes' together with the actual viewing direction.

Another important point in perception management on the object level is handling of occlusions. It is not yet clear, how much of this should be done on the 'object process' level and how much on the situation level; both have to deal with the problem in parallel. On the situation level (upper right corner in fig.3) the semantics in the task context have to be taken into account; on the lower object level the problem is to decide in each evaluation cycle which features belong to which object, how this attribution affects relative state estimation and what is the most likely separation line between the two objects.

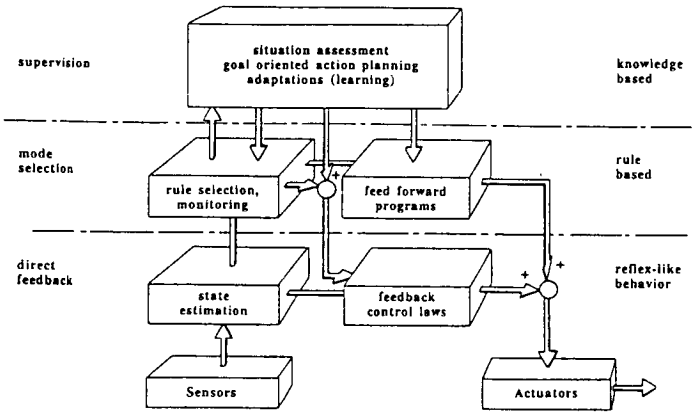
In addition, during this process the question has to be answered which state variables and which shape parameters are presently observable; the interpretation models have to be adjusted correspondingly. Especially the relative viewing angle (in azimuth) of vehicles ahead changes observability frequently in typical highway traffic situations[25]; the problem of vehicle length estimation has already been referred to above.

Intelligent control

The own body carrying the camera is now always represented as an object of the real world (number 1 in fig.3, lower right). Since Newtonian motion is of second order in each degree of freedom the state vector also contains all velocity components in 3-D space; this allows state vector feedback for achieving some goal function in an optimal way. Figure



a) Selectable fast, reflex like feedback control determination with triggered feed forward components; situation dependent control mode



b) Hierarchical scheme for adaptable fast control determination

Fig. 4: Situation dependent, knowledge based intelligent control[28]

4 shows the general scheme adopted in the 4-D approach to intelligent autonomous systems based on state feedback for fast reactive counteraction to perturbations and event-triggered feed-forward control adaptation to a new situation, both managed by knowledge based situation assessment and behavior selection; by purpose, this is not called behavior planning since the generic, well proven behavioral capabilities are available (or may be learned in more advanced versions) and are just invoked with the right set of parameters. The actual control laws, of course, are specific to the task at hand.

The rest of the paper gives a survey on the different application areas to which the 4-D approach has been successfully applied; it is grouped according to the task fields: Rendez-vous and docking, surface vehicle guidance, and dynamical grasping in 3-D space.

Rendez-vous and docking

Most of the methodical developments of the 4-D approach have been performed by Wuensche[29] on the single sensor, single object problem of controlling planar motion in three degrees of freedom of a tabletop aircushion vehicle with reaction jet control relative to another 3-D body (satellite model plant in the laboratory). The real-time system has been controlled by a VAX 750 combined with a custom-made 8-bit image sequence processing system BVV1[2]. The system performed a self-calibration of the horizontal mounting direction of the camera relative to a docking rod for final mechanical fixture. All six state variables relative to the docking partner of known polyhedral shape, and the vertical mounting direction of the camera have been estimated continiuously by tracking four corner features. Which ones of the usually eight visible features should be selected for tracking in order to achieve optimal accuracy of relative position estimated, has been decided by the system itself exploiting the entries into the Jacobian matrix. While the usage of modified Kalman filters has found very wide acceptance in the vision community in the meantime, the more detailed exploitation of the information in the Jacobian matrix does not seem to have been appreciated correspondingly.

A somewhat different type of rendez-vous with one relative state component (horizontal speed) appreciably different from zero (about 200 km/h in the actual example flown) is the landing approach of an aircraft; in this spatial maneuver the number of state variables is doubled to twelve and there are four analog control variables instead of the three discrete ones with the satellite. Large perturbations may occur due to wind gusts; therefore, inertial sensors (rate and position gyros as well as accelerometers) have been important for robust recognition of the relative position to the runway over the last 1 to 2 Km during approach. For initialisation, signals from a Differential Global Positioning System DGPS have been used[30,31]. Thus, the guidance system may be classified as multi-sensor, single object (besides the own body, of course).

The system has been developed over a period of more than a decade[32], starting from simple all-software-simulations. Lateron, in moving base simulations (three rotations) with computer generated imagery and the real sensor and computer hardware in the real-time loop, fully automatic, on-board autonomous landing approaches (including side-winds and gusts) until touch down have been demonstrated; this type of flight simulator for machine vision autopilots seems to be the only one in operation up to now.

Real flight experiments have been performed in 1991 with a twin turbo-prop aircraft D0 128 of the University of Braunschweig; the human pilot was in control, but the perception system estimated the complete state vector at a rate of 16 Hz.

In the meantime, the hardware base for the system has been changed to transputers and a bifocal camera arangement; new flight experiments are planned for spring 94.

Surface vehicle guidance

Contrary to the developments in the US-DARPA ALV program, without having knowledge about these activities at all, we started from a behavioral approach based on the 4-D method for continuous vision processes. No higher level AI-components have been involved on our side initially; the system was capable of recognizing local road environments and of reacting in such a way that certain predescribed behavioral parameters like speed, offset from a line or maximum lateral accelerations were observed. The capability of performing (elements of, or full) missions developed over time on this base.

Local reactions for motion control

Road runner: Taking the guideline model for the construction of high-speed roads as the essential knowledge component for recognizing a road recursively while driving on it, a substantial gain in efficiency for image sequence processing has been realized[33]. Mysliwetz[16,34] extended this to robust road recognition, including the general case of hilly terrain, by applying the 'Gestalt'-idea -known from psychology- to road shape recognition from a large number of (approximate) tangent elements. In the classification scheme discussed, this work till the end of the 80ies belonged to the single sensor, single object group. For more demanding real world applications, especially high speed driving, it turned out that a combination of cameras with both small and large focal length, termed 'bifocal vision', is desirable; this combination has been in use for years, but with separate signal evaluation for different objects and purposes. Recently, the signals from both cameras have been used for recognizing the one object road in a joint evaluation[35] ('two sensors, one object'- case; upper central part in fig.3).

Intelligent cruise control: Adding to this lane keeping capability the one for obstacle recognition, relative state estimation and relative state control[23,24], within the same framework of event-triggered feedback and feed-forward behavior selection as shown in fig.4a, remarkable performance sufficient for driving on 'Autobahnen' in normal traffic situations has been achieved[24].
The *reactive feedback* scheme (lower level in fig.4a) is used

for realizing:

- lane- and speed-keeping with speed adjusted to horizontal curvature such that a preset lateral acceleration level is not exceeded;
- convoy driving behind another vehicle with distance depending on speed driven (2 seconds rule); this includes 'stop & go' driving in traffic jam as a special case.

The *event-triggered feed-forward scheme* (upper level in fig.4a) provides the capabilities for:

- transition from the unconstrained cruise, lane keeping mode to the convoy driving mode (including stop in front of an obstacle);
- lane changing to left and right. (At present, the human driver has to check whether the intended lane is free; in response to an inquiry he triggers the lane change by hitting a key on the board.)

More than 2 000 km have been travelled autonomously in normal Autobahn-traffic since September 1992 with the two vehicles **VaMoRs** of UniBwM and **VITA** of our industial partner Daimler-Benz.

These results on the two lower levels of fig.4b are achieved essentially with differential representations and local considerations; a different situation occurs when global points of view in a task context come into play. The upper level in 4a (the medium one in 4b) forms the transition from strict, local reactive control to global mission performance.

Global performance (mission control)

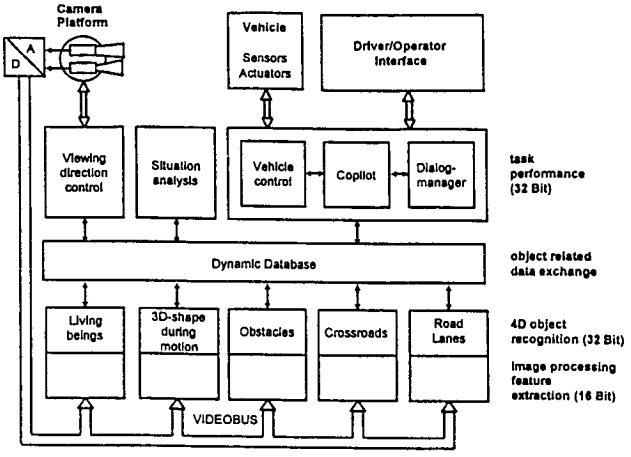
The result of an application of a stereotype feed-forward control time history is a state change with roughly predictable differences between initial and final conditions; this so-called 'maneuver element' may be labeled by a symbol and stands for the finite state transition as an integral representation of this control sequence (e.g. 'lane change': Same driving conditions except for a lateral offset of one lane width).

Maneuver elements may be generic by specifying some parameters which modify the control sequence; for example, lane change may be specified as smooth or rough by fixing the maximum lateral acceleration limit in an otherwise structurally fixed control sequence; this is equivalent to specifying the maneuver time allowed. On the Autobahn, all navigation is done by proper lane changes and lane following; this makes mission performance rather simple. Besides the well structured environment, this was one of the reasons for choosing Autobahn-driving as the first field of application for practical machine vision at the end of the 70ies.

When the capability of reading traffic and navigation signs on the Autobahn is added to the existing system, this goal will be achieved.

Landmark navigation on the factory floor is much more demanding, though in case of failures the damage possible is much less because of the low speeds driven. Taking doors, well visible features on workbenches and other rectangular markers with special height-to-width ratios as landmarks, the suitability of the 4-D approach for real-time visual landmark navigation has been demonstrated in 1991 with moderate computing performance available[36] with an AGV in a laboratory environment and in a factory hall.

Driving on road nets with an automobil in an autonomous mode performing an abstractly defined mission is the most demanding task demonstrated, though yet far from robust real-life applicability. The system shown schematically in fig.5 is in the final stage of development for this



- Bifocal camera pair: tele, wide angle with active viewing direction control
- Object-related, intelligently controlled feature extraction
- Recognition of moving objects exploiting spatio-temporal models (4D)
- Situation analysis in task context (AI-Aspects)

Fig. 5: Transputer based system for autonomous mission performance

purpose: the lower line of blocks is formed by object-specific processor groups each consisting of 16-bit processors for feature extraction and 32-bit processors with floating point units for recursive state estimation; usually, these groups work on separate areas of the image for which they make their own predictions. Occlusions have to be dealt with in cooperation between such processor groups.

All object related data are exchanged via a dynamic data base (DDB) which always contains the most recent estimates of object states and parameters. The higher levels of

the system shown above the DDB incorporate situation- and control- specific knowledge for finding the best behavioral mode in the actual situation. Both fast reacting state feedback control laws and event triggered feedforward control time histories (as discussed for lane changing but also for turning off onto a cross-road) may be applied. A landmark navigation component has been added on top of this for fully autonomous mission realisation[37].

The system, in a different mode of operation, may also be used for monitoring and warning when the human driver is in control of the vehicle[38].

Dynamical grasping in 3-D space

Quite a different application of the 4-D approach has been demonstrated in May of 1993 during the Spacelab mission D2 with the Space-Shuttle Columbia. One of the experiments on board was the RObot Technology EXperiment ROTEX of DLR, Germany; in this set of tasks under the direction of G.Hirzinger one of the tasks was to grasp an object freely floating in space in a confined workcell for safety reasons. The relative position between the object and the six-degree-of-freedom robot arm was to be determined from a camera in the hand of the robot; one of the difficulties was that the computers for visual interpretation and control had to be on the ground. Due to the routing via three geostationary satellites and quite a few groundstation computers the lumped delay time from measurement taking until the control signal derived from these data again arrived on board the Spacelab was around six seconds!

This-time delay has been compensated by exploiting the dynamical models for the object to be caught and for the robot arm. On May 2nd, 1993, this maneuver has been performed by 'remotely-brained machine vision' automatically after initialisation of visual tracking by a human operator[39].

Conclusions

The development of the 4-D approach to dynamic machine vision continues to be successful. Spatio-temporal models oriented towards physical objects together with the laws of perspective projection in a *forward*-mode (and as approximate linear relationships between the states or parameters of the physical objects and the features by which these objects may be visually recognized) are the core elements of the method. The spatio-temporal models as invariants for object recognition also serve for integrating multi-sensory measurement data.

By prediction error feedback an internal symbolic 4-D representation of processes involving these objects is being maintained allowing situation assessment and longer term predictions.

For specific tasks, behavioral capabilities can easily be realised by state feedback or feed-forward control. The internal feedback loops from state prediction to measurement activities in the image plane make interactions between the higher and lower processing levels very efficient.

References

[1] R.E. Kalman: A new Approach to Linear Filtering and Prediction Problems. Trans. ASME, Series D, Journal of Basic Enigneering, 1960, pp 35-45

[2] E.D.Dickmanns, V.Graefe: a) Dynamic monocular machine vision, b) Application of dynamic monocular machine vision. J. Machine Vision & Application, Springer-Int., Nov. 1988, pp 223-261

[3] AGARD Lecture Series 185 'Machine Perception', Hampton, VA, USA; Munich; Madrid, Sept. 1992, Chapter 6, 7

[4] U. Franke, H. Fritz, A. Kühnle, J. Schick: Transputers on the road. Proc. World Transputing Conference '93, Aachen, Sept. 1993

[5] A. Blake, A. Yuille (eds.) 'Active Vision', MIT-Press, Cambridge MA, 1992

[6] K. Kluge: YARF: A System for Adaptive Navigation of Structured City Roads. Ph.D.Thesis, CMU School of Comp. Sci., Febr. 1993

[7] D. Koller, K. Daniilidis, T. Thorhallson, H.-H. Nagel: Model-Based Object Tracking in Traffic Scenes. In Proc. ECCV '92, S. Margherita, Italy. Lect. Notes in Computer Science 588, Springer-Verlag, Berlin,1992

[8] D. Terzepoulos, D. Metaxas: Dynamic 3D models with local and global deformations: Deformable superquadrics. IEEE Trans. PAMI, Vol. 13, No. 7, 1991, pp 703-714

[9] R. Koch: Dynamic 3-D Scene Analysis through Synthesis Feedback Control. IEEE Trans. PAMI, Vol. 15, No. 6, 1993, pp 556-568

[10] H.S. Sawhney, J. Olinsis, A.R. Hanson: Image Description and 3-D Deconstruction from Image Trajectories of Rotational Motion. IEEE Trans. PAMI, Vol. 15, No. 9, Sept. 1993, pp 885-898

[11] J. Weng, N. Ahuja, T.S. Huang: Optimal Motion and Structure Estimation. IEEE Trans. PAMI, Vol. 15, No. 9, Sept. 1993, pp 864-884

[12] T.N. Tan, K.D. Baker, G.D. Sullivan: 3D Structure and Motion Estimation from 2D Image Sequences. Image and Vision Comp., Vol. 11, No. 4, May 1993

[13] B. Sridhar, V.H.L. Cheng, A.V. Phatak: Kalman filter based range estimation for autonomous navigation using imaging sensors. IFAC Symposium 'Automatic control in aerospace, Tsukuba, 1989, Pergamon Press, 1990, pp 45-50

[14] E. D. Dickmanns:2D-Object recognition and representation using normalized curvature functions. In: M.H. Hamza (ed) Proc. IASTED Int. Symp. on Robotics and Automation '85. Acta Press, 1985, pp 9-13

[15] K.D. Kuhnert: Zur Echtzeit-Bildfolgenanalyse mit Vorwissen. Diss., UniBw München, Fakultät LRT, 1988.

[16] B. Mysliwetz: Parallelrechner-basierte Bildfolgen-Interpretation zur autonomen Fahrzeugführung. Dissertation, UniBw München, Fakultät LRT, 1990

[17] Dirk Dickmanns: KRONOS-Users Guide. UniBwM/ Inf./1992

[18] E.D.Dickmanns: Subject-object discrimination in 4D dynamic scene interpretationfor machine vision. Proc. IEEE-Workshop on Visual Motion, Newport Beach, 1989, pp 298-304

[19] P.S. Maybeck: Stochastic models, estimation and control. Vol. 1, Acad. Press, 1979

[20] T.J. Broida, R. Chellappa: Estimating the kinematics and structure of a rigid object from a sequence of monocular images. IEEE Trans. PAMI, Vol. 13, 1991, pp 497-513

[21] H.J. Wünsche: Detection and Control of Mobile Robot Motion by Real-Time Computer Vision. In: N. Marquino (ed) Advances in Intelligent Robotics Systems. Proc. of the SPIE, Vol. 727, 1986, pp 100-109

[22] E.D.Dickmanns, T. Christians: Relative 3D-state estimation for autonomous visual guidance of road vehicles. In: Kanade, T. e.a. (ed.): 'Intelligent Autonomous Systems 2', Amsterdam, Dec. 1989, Vol.2 pp. 683-693; also appeared in: Robotics and Autonomous Systems 7 (1991), Elsevier Science Publ., pp 113 - 123

[23] E.D.Dickmanns, B.Mysliwetz, T.Christians: Spatio-temporal guidance of autonomous vehicles by computer vision. IEEE-Trans.on Systems, Man and Cybernetics, Vol.20, No.6, Nov/Dec 1990, Special Issue on Unmanned Vehicles and Intelligent Robotic Systems, pp 1273-1284

[24] E.D.Dickmanns, R.Behringer, C.Brüdigam, D.Dickmanns, F.Thomanek, V.v.Holt: An All-Transputer Visual Autobahn-Autopilot/Copilot. Proc. ICCV'93, Berlin, May 1993. Also in TAT/WTC'93, Aachen, Sept. 1993

[25] M.Schmid: 3D-Erkennung von Fahrzeugen in Echtzeit aus monokularen Bildfolgen. Dissertation, UniBw München, Fakultät LRT, 1993

[26] W. Kinzel; E.D. Dickmanns: Moving Humans Recognition using Spatio-Temporal Models. XVII-th Congress of the Int. Soc. for Photogrammetry and Remote Sensing (ISPRS), Washington D.C., 1992

[27] J.Schiehlen, E.D. Dickmanns: Two-Axis Camera Platform for Machine Vision. 57th AGARD-GCP-Symposium, Seattle, 12.-15. Okt. 1993

[28] E.D.Dickmanns: 4D Dynamic Vision for Intelligent Motion Control. In C. Harris (ed.): Int. Journal for Engineering Applications of AI (IJEAAI), Special Issue 'Intelligent Autonomous Vehicles Research', Vol.4, No.4, pp.301-307, 1991

[29] H.J. Wünsche: Bewegungssteuerung durch Rechnersehen. Fachberichte Messen, Steuern, Regeln, Bd. 20, Springer-Verlag, Berlin, 1988

[30] R.Schell: Bordautonomer automatischer Landeanflug aufgrund bildhafter und inertialer Meßdatenauswertung. Dissertation, UniBw München, Fakultät LRT, 1992

[31] Dickmanns, E.D.; Schell, R.: Autonomous Landing of Airplanes by Dynamic Machine Vision. Proc. IEEE Workshop on 'Applications of Computer Vision', Palm Springs, Nov./Dec. 1992

[32] G.Eberl: Automatischer Landeanflug durch Rechnersehen. Diss., UniBw München, Fakultät LRT, 1987

[33] E.D.Dickmanns, A.Zapp: A Curvature-based Scheme for Improving Road Vehicle Guidance by Computer Vision". In: "Mobile Robots, SPIE-Proc. Vol. 727, Cambridge, Mass., Oct. 1986, pp 161-168

[34] E.D.Dickmanns; B.Mysliwetz: Recursive 3D Road and Relative Ego-State Recognition. IEEE-Trans. PAMI, Vol.14, No.2, Special Issue on 'Interpretation of 3D Scenes', February 1992, pp. 199-213.

[35] R.Behringer; V. v.Holt; D. Dickmanns: Road and Relative Ego-State Recognition. In: Intelligent Vehicles, IEEE, SAE, Detroit, 1992

[36] C.Hock; E.D.Dickmanns: Intelligent Navigation for Autonomous Robots Using Dynamic Vision. XVII-th Congress of the Int. Soc. for Photogrammetry and Remote Sensing (ISPRS), Washington D.C., Aug. 1992

[37] C.Hock: Wissensbasierte Fahrzeugführung mit Landmarken für automome Roboter. Dissertation, Fakultät LRT, 1993

[38] M.Kopf: Ein Beitrag zur modellbasierten, adaptiven Fahrerunterstützung für das Fahren auf deutschen Autobahnen. Diss., UniBw München, Fakultät LRT, 1993

[39] C.Fagerer, Dirk Dickmanns, E.D.Dickmanns: Visual Grasping with Long Delay Time of a Free Floating Object in Orbit. UniBwM / LRT / WE13 / FB 93-3

AIAA-94-1244-CP

FUSION OF LASER AND IMAGE SENSORY DATA FOR 3-D MODELING OF THE FREE NAVIGATION SPACE

M.Maas, A.Moghaddamzadeh and N.Bourbakis
Binghamton University
T.J.Watson School
Dept. EE, AAAI Lab
Binghamton, NY 13902

ABSTRACT

In this paper a fusion technique is presented for the 3-D modeling of the free navigation space. The fusion technique could be used by an autonomous robot to model and acquire the navigation space and for the extraction of the 3-D map from the environment. The fusion technique is based on the appropriate synthesis of two different sensory data generated by a vision camera and a laser scanner.

KEYWORDS: Fusion, Image Segmentation, Laser Scanning, 3-D images

This work is a part of an FR grant 1992-93

1. INTRODUCTION

Humans and animals are born with multiple senses that allow them to develop a very clear picture of our world. Thus, in the design of autonomous robots, or robust vision systems, the cooperation of the multiple sensors is needed to enhance the system's chances of success in a complex environment. Of particular importance is how the robot formulates a 3-D image of the environment. We intend on using the fusion of a color segmented picture and a laser range finder to develop a working model of the space.

The idea of fusing two sensory signals to formulate a 3-D image is by far not new. Extensive work has been done in this field of machine vision. We encourage the reader to further enhance their background by looking at [1] and [2].

Several very interesting approaches have been done in the field of fusing intensity and range data. Of particular interest is the work done by J.K. Aggarwal et. al. [3].

In particular in [3], a method of fusion between range data and intensity is presented. One major goal of this approach was to minimize the extensive amount of time required in sensing the range data. Using potential points of interest from the intensity data as directors for the range sensor, the range sensor senses those points of interest and then the range data and intensity data are combined to form the graph.

Another method is discussed in [4]. This method uses the most dominant sensor at one time (range or intensity) for seed segmentation. This segmented area then again uses the data from the range and intensity for region merging. The authors used this method to minimize the lighting difficulties and color variance (intensity image segmentation), and to minimize the difficulties range image segmentation has with sloped surfaces.

Other related work is available in [5],[6], and [7].

It is interesting to be mentioned that the laser scanner has very high resolution (similar to a digital image) very close to the scene surface. This means that in longer distances, such as 1 m or longer than that, the resolution changes drastically. For instance, in a distance of 1 m, the size of the laser spot has diameter of 1 cm approximately. Thus, how the image 3-D model will be developed under these conditions.

We propose a methodology, which hopefully will correct some of these problems, such as long distance 3-D image modeling and color difficulties, by using the intensity data as the director for the laser. More specifically, we use a fuzzy image segmentation technique [9] for the sufficient separation of the image regions with different color. In addition, we generate an image with a low resolution equivalent to the laser one, thus the fusion of these sensory data to be possible and the final 3-D image to contain less "noise" on the scanned surfaces.

This paper is organized into five sections. Section 2 discusses the laser range and some of the existing problems. Section 3 presents briefly the fuzzy image segmentation technique for the separation of the regions. Section 4 describes the fusion methodology of these two different sensory data. The last section summarizes the overall presentation.

Copyright © 1993 American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

2. LASER RANGE DATA ACQUISITION

Ideally we would like to scan the picture area with a laser scanner. This would provide us with a range map that could be "fused " with our intensity segmented map. However the inhibitive high cost of laser scanners is not within our university's budget.
That leaves us with two options; we can attempt to build our own laser scanner or we can purchase a point by point laser range detector that can be manipulated in a xy plane. Due to the time constraints of this paper we will assume that we have the funds to purchase a point by point detector. It is also worth noting that it would probably take us a considerable amount of time to build the complex circuitry required for a laser scanner. Our objective is to develop a new fusion technique.

One company that has a variety of laser distance meters is RIEGL[8]. Features included in these detectors are RS233 interface (as long as measure times are > 50 ms), immunity to electro-magnetic and acoustic interference, high speed and accuracy, and a starting beam width of ≈ 1cm with a divergence of 3.2 mrad. Since our robot is designed primarily for enclosed hazardous environments the laser sensor range does not need to be exceptionally long. The meter fulfilling our needs the best is the LD90-210-CX/C6. It's sensing range is from 1 meter to 15 meters. In addition [8] also has a biaxial rotary mount (BD 90). The mount enables us to selectively pick the intensity segmented areas.

Theoretically a beam width the size of a pixel would be ideal. This would permit a one to one fusion with the pixels form the intensity segmentation. However technology has not yet reached this minute beam width. The authors of this paper are aware of very small diameter beam widths but these lasers are not yet available.

After the separation of the intensity scan into specific colors the area of each color segment is determined and put in xy coordinates. Once an area is defined the bidirectional mount is manipulated so that the laser range meter can determine the range of the area in question. When the scanning of the area is complete another color area is scanned until the entire intensity frame has been scanned. The intensity map and the range map are then fused to form our first attempt at the 3-D image.

One of the critical features of the laser scanner is the non-uniform scanning of the scene. More specifically, since the mechanical XY movement mechanism used by the laser scanner is imperfect, there is the possibility that the laser beam to overlap two consecutive scanned point, or to be far away from each other, as shown in figure 1. A solution to this particular problem is the scanning of the same scene area twice, and the averaging of two distance values by reducing the possible error.

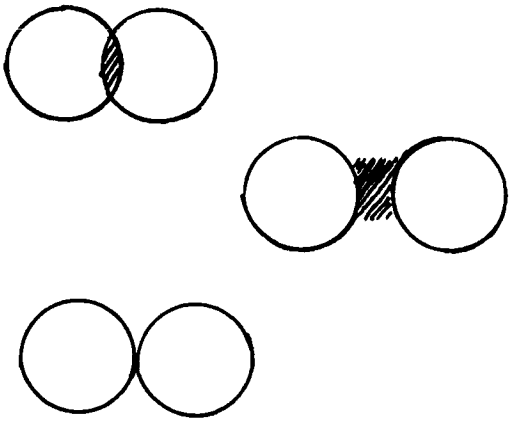


Figure 1: The imperfect scanning

3. FUZZY IMAGE SEGMENTATION

Segmentation and edge detection in color images are not deeply investigated in the literature. Some methods separate the three color components and work on each independently. Three different histograms are drawn and after segmentation or edge detection, the results are combined to form segmented or edge detected color images. The results using these techniques are false because of three reasons: one, segmentation or edge detection based on color alone is not complete. The relative position of the pixels and their color in their neighborhood are important as well. Two, in real images there is no sharp valley in histograms. Three, once one pixel color is mapped to three locations on three histograms, pixel color information is scattered. And once this is done for thousands of pixels, all available information is scattered and mixed. After thresholding doing the reverse and putting the three components back together is not an easy job. This technique is usually used for images having a limited number of segments or clusters.

The technique described in this section is applicable to both color and gray level images having an unlimited number of clusters. A cluster is defined as a collection of touching pixels which have almost the same color or the change in color is gradual. Since an unlimited number of clusters could be detected from an image, displaying each cluster in a different image would not be practical. What we have done is displaying all clusters in one image. Each cluster is shown using a color that reflects the main color of the cluster.

Before segmentation can begin a rough edge detection must be performed. The used edge detection technique is not described here but it is sensitive enough to detect all sorts of edges (crisp, fuzzy, thin, and week). A histogram table is also generated and sorted according to the number of pixels. The first three entries of the table for a particular image are shown below:

21 27 23 892
21 26 23 640
8 12 28 591

The numbers represent red, green, blue, and the number of pixels in the image having that particular color vector, respectively. Segmentation is performed in the following steps: 1- find big clusters, 2- expand clusters, 3- find medium size clusters, 4- expand clusters, 5-find small clusters, 6- fill in the blanks.

Big clusters are defined as those having a minimum of 150 pixel members. To find big clusters instead of looking for valleys in the histogram we look for peaks. The first peak is the first entry in the histogram table. Therefore, the seed color is known and the seed location must be found. The image is scanned for that particular color vector and once found it is test-grown and pixels marked until a detected edge is reached and the size of the test-grown cluster is noted. This is done for all unmarked pixels having the seed color. The seed location that uses the most of the seed color pixels and is big is chosen as the seed location and its color as the cluster color. The seed is then grown. While the seed is growing and pixels are added to the cluster, each pixel is marked and its entry in the histogram table decrement. Meaning, that particular pixel is not available any more and does not contribute to the histogram table. When the seed is all grown, the histogram table is resorted and the first entry, now, is the second peak of the histogram. This growing process is repeated until there are no more big clusters left.

Since the edge detection algorithm used is very sensitive to variance, the detected edges are thick especially in the case of fuzzy edges. Therefore, big clusters must be expanded using fuzzy logic to fill out that gap before smaller clusters are found. Big cluster expansion is performed in two steps: In the first step each cluster is expanded (surrounding pixels added to the cluster) based on two rules:

1. The absolute variance (difference between the pixel's color and the seed/cluster's color) is low. and
2. The relative variance (difference between the next and the previous pixel's color) is low. This is used to include gradual changing shade areas as well.

In the second step each cluster is expanded based on the following rule:

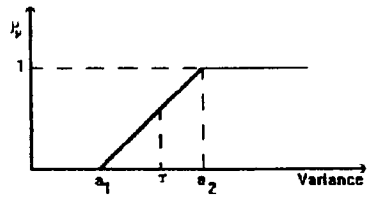
For each pixel the absolute variance with the surrounding clusters and the distance (in number of pixels) with that cluster is evaluated and their product recorded (degree of farness). If the lowest product value (which is for the closest cluster in terms of color and distance) is lower than a threshold, the pixel is added to the cluster.

The following equation is used to calculate variance between pixel 1 and 2.

$$Variance_{1,2} = (R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2$$

where R, G, and B are the three color components. Variance membership function is shown below:

$$\mu_{V_{1,2}} = \begin{cases} 0 & if\ Variance \le a_1 \\ 1 & if\ Variance_{1,2} \ge a_2 \\ \dfrac{Variance_{1,2} - a_1}{a_2 - a_1} & else \end{cases}$$



Averaging is used in anding the two fuzzy membership functions.

Medium size clusters are found the same way big ones were but this time they are grown based on the first step of expansion rules (minimum size 30 pixels total). They are then expanded based on the second step of expansion rules.

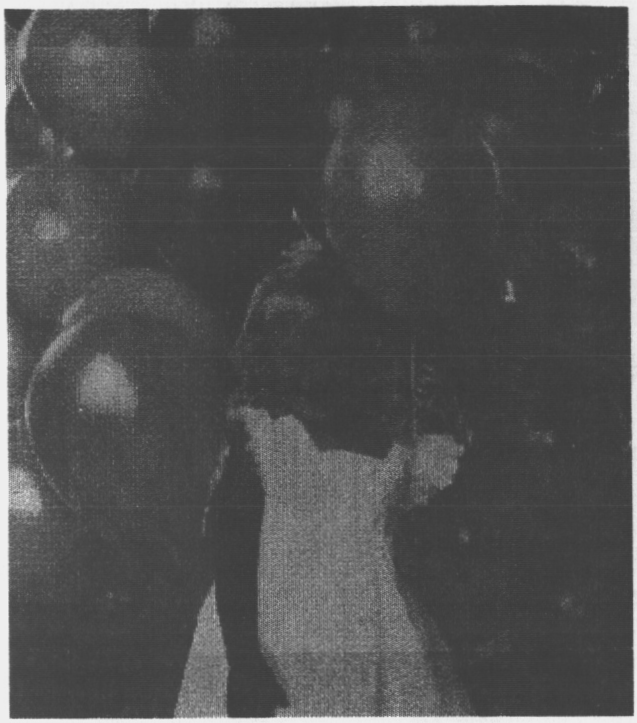
To find small clusters, for each undecided pixel in the image the degree of farness with the nearby clusters is evaluated.

If the lowest value is higher than a threshold, use the pixel as a seed and test-grow it based on the first step of expansion rule. If the test-grown cluster has at least 6 pixel members and has a big contrast with the closest cluster (its existence is of importance), grow it, otherwise leave it undecided. To fill in the blank for all undecided pixels the closest cluster (in distance and color) is found and the pixel is added to that cluster. Figure 2 shows a colored image and its segmentation. Figure 3 shows a grey image followed by its segmented one.

4. THE FUSION METHODOLOGY

4.1. Methodology

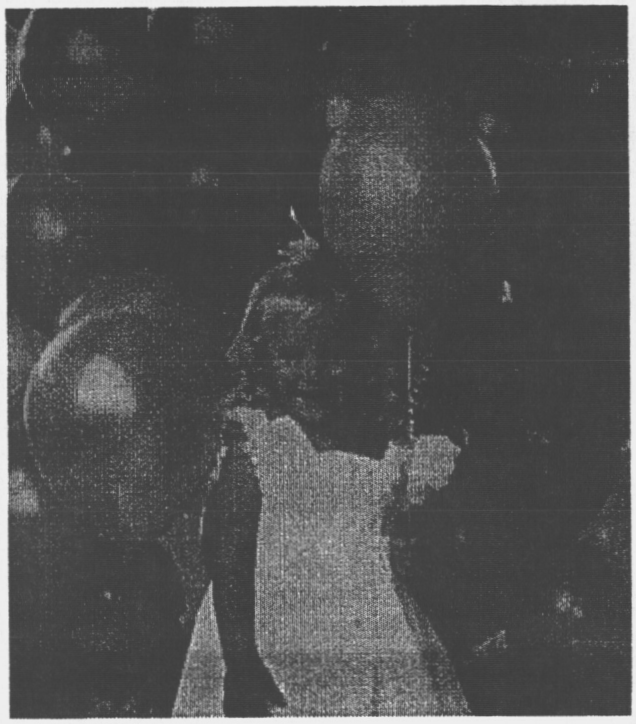
The fusion methodology presented here is based on the synthesis of different sensory data generated by a vision system and a laser scanner. In particular, the camera receives an image from the environment. The digital form of the image P(i,j), 1<i,j<n, where "n" represents the size of the image matrix, is stored in the main memory. The digital image represents a 2-D array of gray level values or color values (gv(i,j)) for the image pixels. At the time that the image is grabbed by the camera, the laser scanner "scans" the same area of the



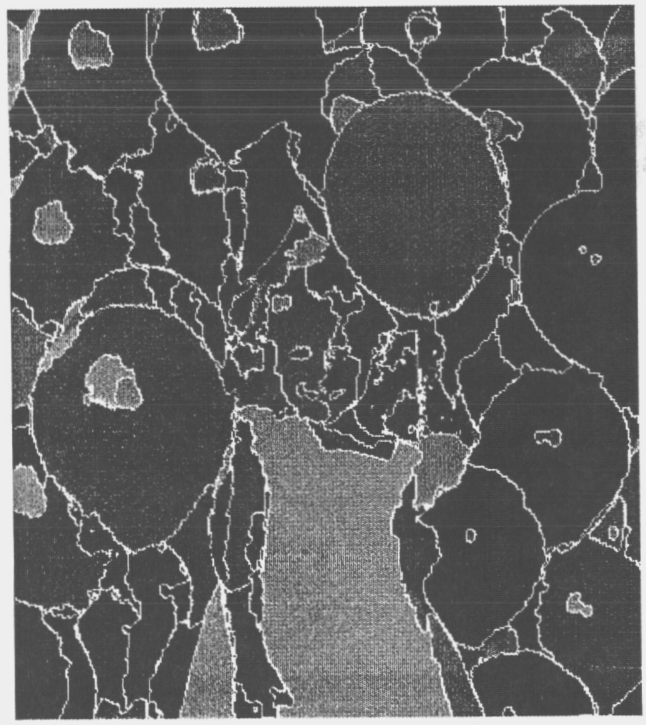
(a)

(b)

Figure 2: a) A colored image
b) Its segmentation



(a)



(b)

Figure 3: a) A grey level image
b) Its segmentation

environment by producing an 2-D array D(k,m). The values d(k,m) of the D array represent the distances of the laser scanner from the surface of the current navigation space at the particular area from which the image was grabbed. The D array is also stored in the memory. From this point, the fusion technique takes place. More specifically, a fuzzy segmentation technique is applied on the image values to define the regions with "color" similarities. An edge detection technique is also applied on the same image to define and separate the boudaries of the image regions [9]. Since the resolution of the P image-matrix is higher to the D distance-matrix, the resolution ratio (R(F)= [R(P)/R(D)]) is calculated. This R(F) ratio will guide the image pyramid generation technique for the selection of the pyramidal level of the image (PL) with the closest resolution to the D matrix, figure 4.

At this point, the fusion matrix F is generated by combining the coordinates of these two matrices PL and D respectively. More specifically, the elements of the F matrix have four values: the x' and y' coordinates, the d distance of these x'y' coordinates, and the gv grey level value which cooresponds to x'y'; f(x',y',d,gv). The 3-D image model (F3) for the original image will be generated by the F matrix with the expansion of each f(x'y',d,gv) element by R(F) pixels. More specifically,

$$f(x',y',d,gv) \longrightarrow S\{p\}$$

where S represents the set of 4^q neighbor pixels, so that $R(F) \lesssim 4^q$. Thus, 4^q neighbor pixels of the 3-D image model will share the same distance d; p(i+w,j+z,d,gv), where $w,z \in \{1,2,\ldots 2^q\}$.

At this point, the colored image regions separated by the segmentation method will be used for "smoothing" the possible unevenness of the 3-D model. More specifically, there will be cases where the pixels, which belong to different colored regions, share the same distance d. In this case, the distance d(Rc) which represents the majority of the pixels in the region Rc, with color "c", will be used to redefine the distance of those pixels, which share a distance different than d(Rc).

4.2. Illustrative Example

In this subsection we present graphically the fusion methodology by using a simple grey level image, figure 5, the reduced image using the pyramidal process, figure 6, a generated distance matrix with equivalent resolution, figure 7, the fused 3-D image at the reduced size, figure 8, and the 3-D model of the original image, figure 9.

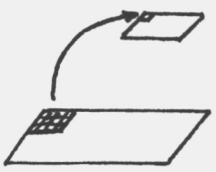


Figure 4: The pyramidal reduction of an image

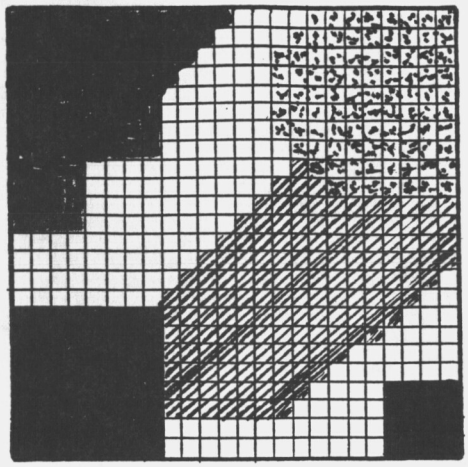


Figure 5: A grey level image

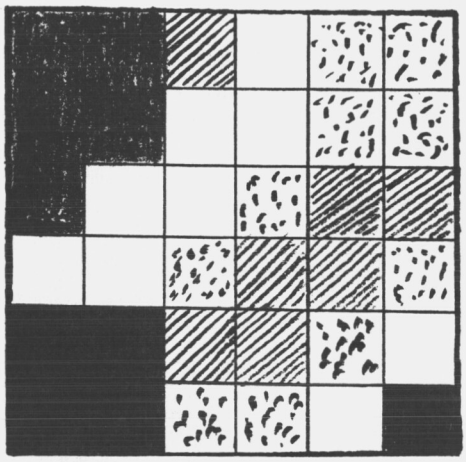


Figure 6: The reduced image using pyramidal procedure.

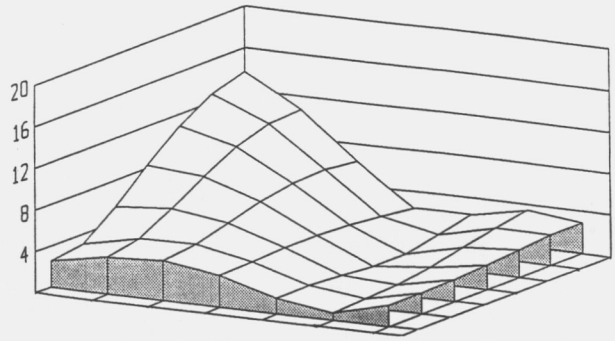


Figure 7: The distance matrix (in a 3-D form)

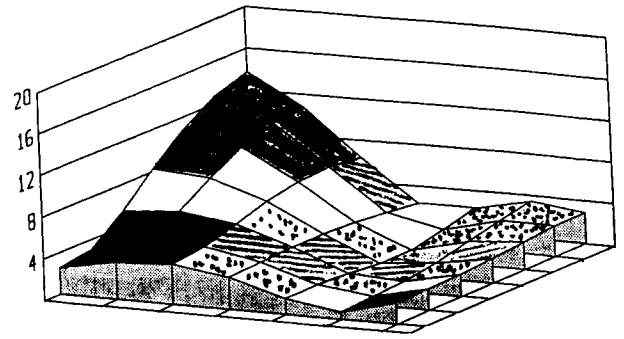


Figure 8: The fused image at the reduced size

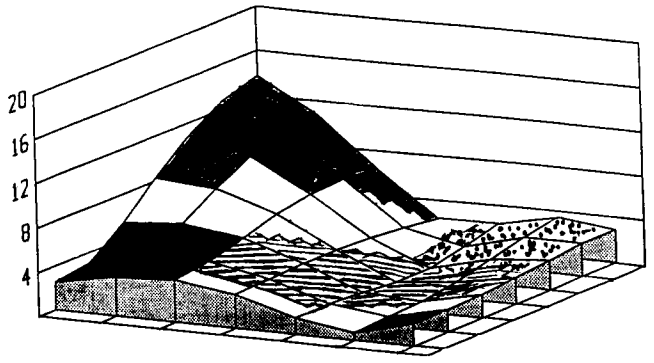


Figure 9: The 3-D model of the original image

5. CONCLUSIONS

In this paper a fusion methodology has been presented. This methodology combines two different type of sensory data for the generation of a 3-D image model. The sensory data used by this method were the distances produced by a laser scanner and the pixels average intensities (or color) of a 2-D digital image grabbed by a vision camera. The problem of the different resolutions for these two sensory data was solved by the reduction of the image resolution, the fusion of the different data and the reconstruction of the original image by using a fuzzy image segmentation technique. Since the laser scanner was available for this experimental work, only feasibility simulated results were presented.

REFERENCES

[1] J.K. Hackett and M. Shah, "Multi-Sensor Fusion: A Perspective," *IEEE International Conference on Robotics and Automation*, pp. 1324-1330, 1990.

[2] J.K. Aggarwal and A. Mitiche, "Multiple Sensor Integration/Fusion Through Image Processing: A Review," *Optical Engineering*, 25:380-6, March 1986.

[3] M.J. Magee, B.A. Boyter, C.H. Chien, and J.K. Aggarwal,"Experiments in Intensity Guided Range Sensing Recognition of Three-Dimensional Objects," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, pp. 629-637, Nov. 1985.

[4] J.K. Hackett and M. Shah, "Segmentation Using Intensity and Range Data," *Optical Engineering*, 28:667-674, June 1989.

[5] B. Gill, A. Mitiche, and J.K. Aggarwal, " Experiments in combining intensity and range edge maps," *Comp. Graph. Image Pro. 21*, 395-411 (1983).

[6] M. Hebert, "Outdoor scene analysis using range data," *IEEE Conf. on Robotics and Automation 3*, 1426-1432 (1986).

[7] R.O. Duda, D. Nitzan, and P. Barrett, "Use of range and reflectance data to find planar surface regions," *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-1, 259-271, 1979.

[8] *RIEGL LASER MEASUREMENT SYSTEMS, RIEGL USA INC.*, 4419 Parkbreeze Court, Orlando, Florida 32808

[9] A.Moghaddanzadeh and N.Bourbakis,"A fuzzy image segmentation-edge detection technique", TR-1993, sub.

AIAA-94-1245-CP

TELE-ASSISTANCE FOR SEMI-AUTONOMOUS ROBOTS

Erika Rogers and Robin R. Murphy †
Dept. of Computer and Information Science
Clark Atlanta University
223 James P. Brawley Dr. at Fair St.
Atlanta, GA 30314
erika@pravda.gatech.edu

†School of Mathematical and Computer Sciences
Colorado School of Mines
Golden, CO 80401-1887
rmurphy@mines.colorado.edu

Abstract

This paper describes current work on a cooperative tele-assistance system for semi-autonomous robots. This system combines a robot architecture for limited autonomous perceptual and motor control with a knowledge-based operator assistant which provides strategic selection and enhancement of relevant data. The design of the system is presented, together with a number of exception-handling scenarios that were constructed as a result of experiments with actual sensor data collected from two mobile robots.

Introduction

Traditional telerobotics research focuses on the separate contributions of the human, computer and robot in the performance of a particular task. Historically a single human at a local system is dedicated to operating a single remote robot. However, continuous supervision may be impractical for applications where the communication bandwidth acts as a bottleneck, and/or transmission time delays make remote high-dexterity control difficult or impossible. Further, the operator may not be reliable due to fatigue, while increased environmental complexity, use of multiple sensing modalities, or the addition of more robots all may contribute to the cognitive overload of the operator.

One approach to these problems is to increase robotic autonomy through the addition of intelligent capabilities. Control schemes, such as *shared control* and *supervisory control*, reduce both the amount of communication between local and remote, and the demands on the operator by increasing the autonomy of the remote. However, there is still a need for human problem-solving capabilities, particularly to configure the remote for new tasks and to respond to unanticipated situations. Thus the teleoperations community is becoming increasingly interested in computerized assistance, both for the effective filtering and display of *pertinent* information or data, and also for the decision-making task itself.

Copyright © 1993 American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

The purpose of this paper is to describe current work on a cooperative tele-assistance system which combines the autonomous perceptual and motor control abilities of the Sensor Fusion Effects (SFX) architecture [4] with the intelligent operator assistance provided by the Visual Interaction Assistance (VIA) system [5]. In this approach, the remote system consists of a robot with sufficient computerized intelligence to function autonomously under a particular set of conditions, while the local system is a cooperative decision-making unit that combines both human and machine intelligence. The computerized aspect of each system enables communication to take place in a common mode, and in a common language.

The paper presents an overview of the design of the system itself, and then discusses a number of experiments and scenarios using data from two mobile robots. The scenarios demonstrate how the local tele-VIA system would assist an operator to respond to a sensing problem which could not be resolved by the exception handling mechanism of the remote robot.

Background and Related Work

Coiffet and Gravez [2] suggest that "instead of searching for an overall model including complex concepts as human behavior, it is more profitable to consider the computer role as performing assistance functions to humans". They go on to describe three situations in which conceivably computer and human can collaborate, and compare the differences between *telepresence*, "a system-oriented assistance centered on machine transparency", and *teleassistance*, which refers to "task-oriented assistance". They state, further, that:

> Increasing the capabilities, and therefore the complexity, of a teleoperation system soon results in a cognitive overload for the human operator. The design of a cooperative system should thus introduce strategic assistance forms that facilitate on-line symbolic

control. Generally speaking, the basis for such assistance is the selection and processing of relevant data (sensor outputs and execution reports) and the filtering of operator commands. This is reflected in a high-level structural dialogue embodying task-oriented diagnosis and proposing pertinent solutions. At the symbolic level, the computer contribution is deduced from an understanding of the work at hand, and is intended to enhance the human operator's decision-making process and to improve on-line human-machine communication.

The work presented in the remainder of this paper is in keeping with the philosophy behind these comments, and demonstrates how such a cooperative system may be designed and implemented.

Approach

In remote and unexplored environments where unanticipated events are likely to occur, it is important that the robot be able to handle a number of situations autonomously and in real-time. In the event of an unresolved problem, however, the remote robot must have recourse to a local human operator for assistance. However, in order for the operator to perform diagnosis effectively, the data from the robot must be appropriately selected and presented. Once the problem has been determined, the operator should then be able to re-configure the robot so that the autonomous processes may once again take over. An important consequence of this approach is that it is now conceivable for the operator to supervise more than one robot simultaneously, and furthermore, to request data from other robots working with the one in trouble.

To achieve this goal of cooperative tele-assistance, two major software systems have been joined together and modified appropriately for this application domain. The first is the Sensor Fusion Effects (SFX) architecture [4], which utilizes state-based sensor fusion to support the motor behavior of an autonomous robot. If a state failure occurs, fusion is suspended and control is passed to an exception-handling mechanism, which attempts to identify the problem and either repair or replace the sensing plan. The second system, called VIA (Visual Interaction Assistant), is designed to cooperatively assist human perception and problem-solving in a diagnostic visual reasoning task. VIA is a blackboard-style system which utilizes knowledge-based techniques to focus the user's attention on relevant parts of the image, automatically enhancing the image according to the needs of the user's problem-solving process. It further manages diagnostic hypotheses, maintaining beliefs according to current evidence, and assists the user to converge opportunistically on a solution where possible.

The advantage of linking the SFX system with the VIA paradigm is that under SFX, the robot has already attempted a certain amount of trouble-shooting itself. Thus information about what has been tried, the robot's own conclusions, and the relevant sensor images can all contribute to the decision-making process of the local operator. In order to achieve this, the teleSFX system includes an interactive exception handling component, which allows the robot to call the operator for help in the event that its own exception handling capabilities could not resolve the problem.

The central communication mechanism between the remote and local intelligent systems is the blackboard structure. The advantage of this architecture is that it allows asynchronous communication between a number of independent knowledge sources. In the general VIA system, the user is considered to be a knowledge source as well, cooperating with the knowledge-based system in the search for a solution (or partial solution). In the case of the tele-assistance system (teleVIA), the remote robotic system is also incorporated as a knowledge source, thus allowing the three entities, robot, teleVIA system, and human operator, to make contributions to the solution of the problem in a cooperative manner. This is especially important in cases where the solution set is not initially tractable, and more information must be acquired in order to quickly constrain the list of diagnostic hypotheses, so that a repair plan may be constructed. An overview of the entire system is shown in Figure 1, and further details are provided in the following subsections. In this diagram, it can be seen how the interactive configuration and interactive exception handling components of the teleSFX architecture are merged with the intelligent assistance provided by teleVIA, through the panels of the blackboard. The emphasis in this paper is on the interactive exception handling aspects of this design.

TeleSFX

In [3], the teleSFX control scheme was introduced, emphasizing the intelligent exception handling mechanism at the remote. Unlike configuration, exception handling must be done in real-time (for example, a robot may be moving when a sensor malfunctions). As shown in [1], autonomous exception handling is difficult because it involves domain and hardware specific information which may not always be available or correct.

TeleSFX uses a three part strategy for exception handling: *detection, classification, and recovery.* The first step, detection, determines that a "sensing failure" has occurred. Sensing failures are any anomalous or suspect conditions that have been previously defined by the knowledge engineer. Sensor malfunctions are one type of failure. Most sensor malfunctions manifest themselves via explicit hardware errors communicated to the controlling process (e.g., bus errors, frame grabber errors) and tend to be straightforward to classify and recover from (e.g., reset the system, request a retry). Another class of sensing failures is due to unanticipated changes in the sensing environment which degrade the performance of one or more sensors (e.g., the lights are turned off). The third and final class of failures stem from errant expectations, where the robot is interpreting the observations according to a model. If for some reason the robot has selected the wrong model at the wrong time (e.g., for mechanical reasons, the robot did not rotate fully to

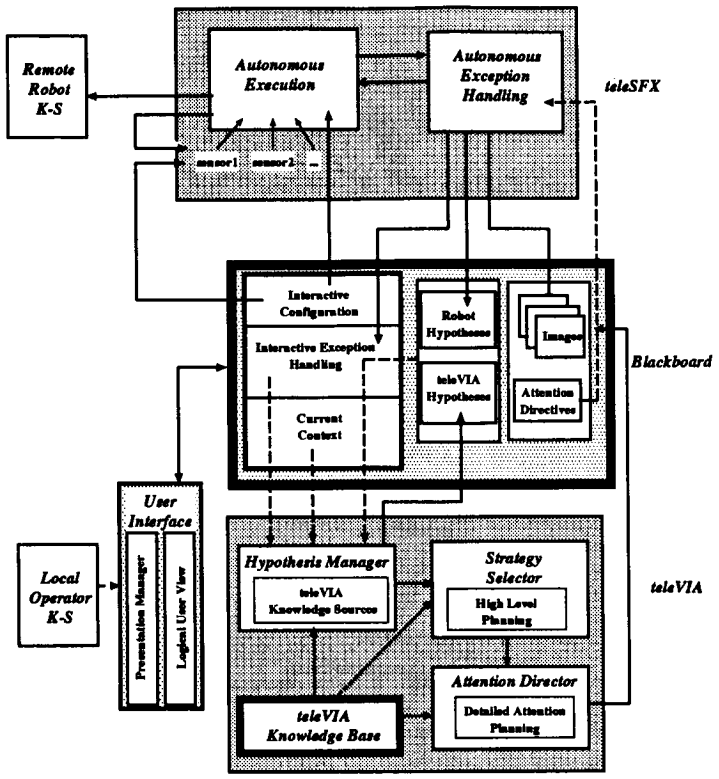


Figure 1: Cooperative Tele-Assistance System Design.

the intended viewpoint), the sensor observations are unlikely to agree.

Failures in the latter two classes are difficult to detect because the sensors are operating "correctly" but their data can no longer be interpreted without accounting for the changed context. Therefore teleSFX is sensitive to inconsistencies in the evidence contributed by different sensors for a particular task. The knowledge engineer defines a set of failure conditions representing these inconsistencies for the particular implementation. Each perceptual process may have a different set of thresholds for those failure conditions, given the unique interactions between sensors.

The classification step has the remote attempt to autonomously identify a sensing failure, and adapt the sensing configuration. This involves hypothesis generation, testing and response heuristics at the remote site, and several experiments have been described in [1] which demonstrate this capability. However, the success of the classification step depends on the expert understanding of the domain and the sensors. This domain-dependence means that classification by the remote is brittle and will not always be successful. Therefore, if the remote system cannot resolve the difficulty, teleSFX must post the request for help to the blackboard, together with immediately relevant data such as current sensor data and a log of the remote's hypothesis analysis. This signals the teleVIA system to activate its knowledge sources in order to request and present further data, as well as to perform further hypothesis analysis.

Figure 2 shows the details of the control system for the remote site. The local operator is involved primarily in interactive configuration, and general monitoring, until the interactive exception handling is triggered by the remote system. At that point, teleVIA takes over from teleSFX until the repair is communicated.

Blackboard

The Blackboard is where the evolutionary results of the problem-solving effort are captured. The original logical partitioning of the blackboard was based on components of a cognitive model of visual interaction described in [5], and was designed to facilitate transfer of information between human perception and problem-solving during a visual reasoning task. In the domain of tele-assistance, it is seen that, with one exception, the same logical partitions or panels may be used. The additional information which is contributed by the remote robotic system is accommodated in the subpanels as shown in Figure 3.

Context Panel

In the general VIA design, this area contains information that is known about the overall problem context. In the teleVIA mode, the Context Panel is used to monitor the robot's (or robots') current activities. It is divided conceptually into three subpanels:

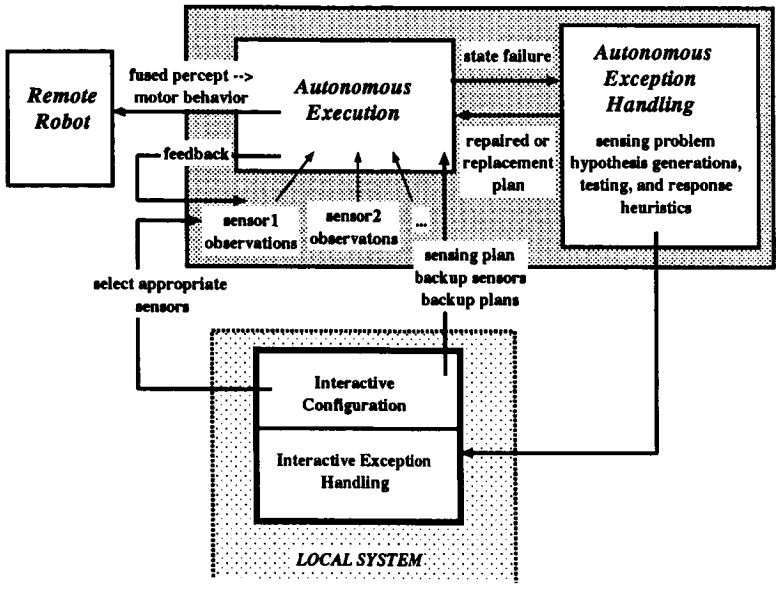


Figure 2: Overview of TeleSFX.

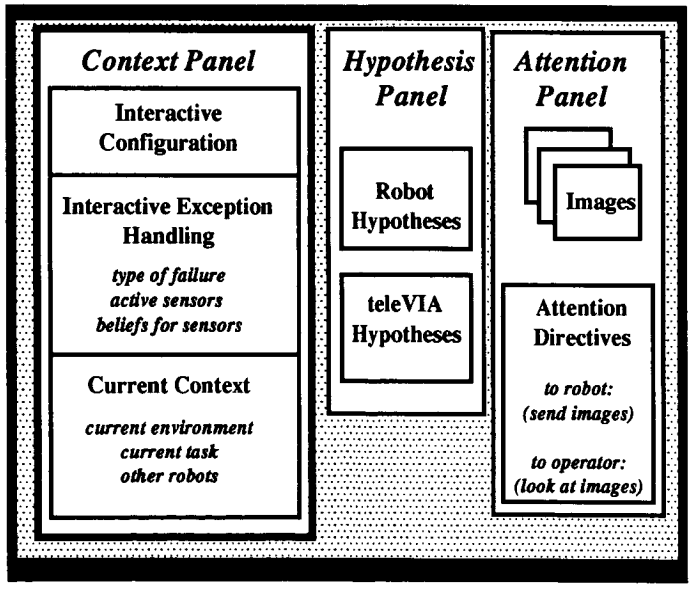


Figure 3: Tele-VIA Blackboard.

1. *Interactive Configuration* allows the local operator to select appropriate sensors, and to communicate sensing and backup plans to the robot.
2. *Interactive Exception Handling* receives the signal for help when autonomous exception handling fails. The remote system immediately posts the type of failure, currently active sensors, and the belief table for those sensors. This tells the local operator what the perceptual status of the robot is at the time of failure, and provides initial information for teleVIA to begin formulating hypotheses, and requesting further information.
3. *Current Context* is a panel which is active during both interactive configuration and interactive exception handling. It contains information about the task underway, the known environmental factors and conditions, which sensors are active and working, and intermittent video images from the robot reinforcing the operator's knowledge of the context within which the robot is currently functioning.

In the initial design of teleVIA, the overall Context Panel is used simply as an informational tool for monitoring the robot.

Hypothesis Panel

This panel contains the current hypotheses that constitute the partial (or complete) solutions that are evolving as a result of the problem-solving activity. It is divided into two subpanels:

1. *Robot Hypotheses* contains the hypotheses generated by the teleSFX system at the remote site, and reflects the diagnostic and problem-solving activities carried out autonomously by the exception handling mechanism of the robot.
2. *TeleVIA Hypotheses* contains the hypotheses generated by the knowledge sources of teleVIA, based on the information posted by the remote system in combination with more extensive knowledge retrieved from the teleVIA knowledge base.

Attention Panel

This panel is the locus of the visual focus-of-attention mechanism. It is also partitioned into two parts:

1. *Attention Directives* are issued by the teleVIA system in order to assist the local operator's perception of relevant data. To accomplish this, teleVIA may request particular images to be transmitted by the robot. In this way, delays due to transmission of unnecessary and/or extraneous data may be avoided. Furthermore, since the images are selected by teleVIA's knowledge sources according to the current problem, they are more likely to be pertinent and useful. The directives issued to the operator are then aimed at guiding him/her to look at particular aspects of the data provided by the remote system.
2. The second area of the Attention Panel consists of one or more images, obtained from the robot by the teleVIA system. Depending on the sensory modality of the displayed images and/or data (e.g., video vs. infra-red vs. ultrasonics), teleVIA will also automatically execute appropriate image enhancements designed to facilitate the operator's perception of the feature(s) in question. In this manner, the superior perceptual capabilities of the human operator can be exploited in order to diagnose the problem more quickly.

TeleVIA

In Figure 4 are shown the components of the cooperative system which assists the human supervisory activities at the local site. TeleVIA contains four main control modules: Hypothesis Manager, Strategy Selector, Attention Director and User Interface, together with a knowledge base which serves as the repository of long-term information in the system. The Hypothesis Manager impacts the blackboard through the activities of hypothesis-related knowledge sources. The Strategy Selector is used to pass control from the Hypothesis Manager to the Attention Director, since the way in which attention is focused may depend on the strategy used for reducing the list of active hypotheses. The Attention Director is concerned with focusing attention by presenting and enhancing images as well as suggestions to the operator of what to look at next. The User Interface is the component through which the human operator communicates with the teleVIA system.

Hypothesis Manager

The purpose of the Hypothesis Manager is to percolate information through the levels of the blackboard via the activities of the knowledge sources. Each knowledge source has a set of preconditions that must be satisfied by information at a particular level of the blackboard. It then performs a transformation of the information at one or more levels. Some examples of knowledge sources which are activated by the type of sensor involved in the failure are illustrated in the following tables.

K-S 1
Precondition: The sensor involved is video.
Action(s): Request latest image from robot. Post image. Retrieve and post video hypotheses.

K-S 2
Precondition: The sensor involved is infra-red.
Action(s): Request latest image from robot. Post image. Retrieve and post infra-red hypotheses.

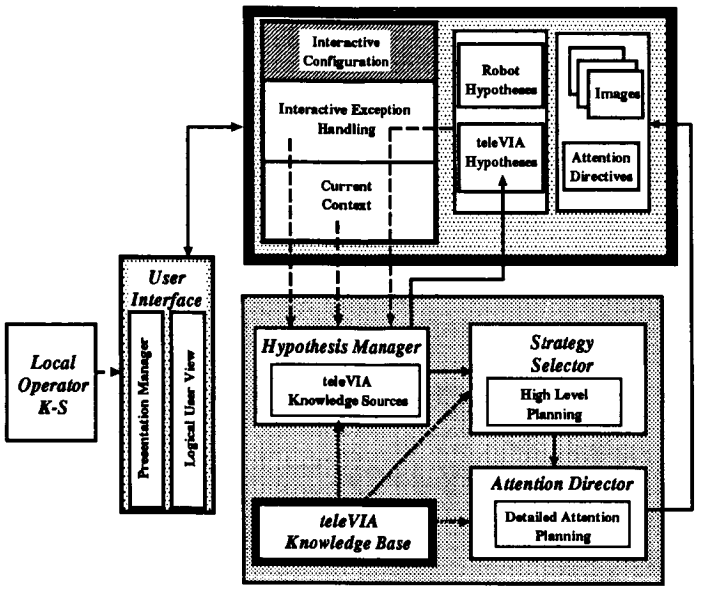


Figure 4: Overview of TeleVIA.

K-S 3
Precondition: The sensor involved is infra-red and image is posted.
Action(s): Retrieve model from current context. if model = available then invoke model-false-color enhancement else invoke generic-false-color enhancement. Post image.

Strategy Selector

This module is invoked by the Hypothesis Manager when a knowledge source needs further information before proceeding. It examines the current blackboard configuration in order to determine an appropriate strategy for the next step in the problem-solving process. A High Level Plan is then generated to carry out the selected strategy, and is passed to the Attention Director for refinement and execution.

Attention Director

The Attention Director module takes the High Level Plan produced by the Strategy Selector, and constructs an Attention Plan that contains detailed instructions for focusing attention. The steps of the Attention Plan are based on the particular type of evidence that is needed to fulfill the mandate of the Strategy Selector. These steps are expanded with image enhancement procedures where appropriate, and are executed. Control is then passed to the operator for feedback. In this way, the system presents information, makes suggestions, and enhances the image(s) in such a way as to influence the direction of the operator's problem-solving.

User Interface

The User Interface is divided into two parts: the *Logical User View*, which controls how much of the blackboard is visible to the user, and the *Presentation Manager*, which controls the form of the interface as it is presented to the user. The Logical User View component of the user interface allows the system to be adapted for various purposes without compromising its basic problem-solving approach. For example, when the operator is simply monitoring the robot, and performing interactive configuration, the panels involved in exception handling should be hidden from view. There may also be a certain amount of data posted to the blackboard, which is utilized by teleVIA in its hypothesis management, but which should not necessarily be visible to the operator. On the other hand, the Presentation Manager provides the actual human-machine interface of the system through a displayed representation of the Logical User View. This may take a number of forms including menus, icons, graphics, and/or direct manipulation windows, and may even extend to audio as well as visual mechanisms.

Experiments

The experiments described here use data for scene recognition which was collected from two sources. Scenarios 1, 2, 3 and 5 are based on sensor observations collected from the Denning DRV mobile robot, George, at the Georgia Institute of Technology. Scenario 4 is based on sensor data from Clementine, the Colorado School of Mines' Denning MRV-3 mobile robot. It should be noted that some of the data has been used in previous experiments. The exception handling activities at the remote in the experiments described in this paper differ from previous work [3]

because they make use of the more rigorous system developed in [1].

Five different types of sensors (an Inframetrics true infrared camera, a black and white video camera, a Hi8 color camcorder, a UV camera and ultrasonics) provided observations from George. Clementine supplied data sets from three sensors (a black and white video camera, a color camcorder, and ultrasonics). Both robots simulated security guards, where the task was to determine whether a student desk area of a cluttered room had changed since the last visit. In the following scenarios the focus is on the activities of the teleVIA system in response to the request for help from the remote system.

Scenario 1

In the first experiment, the robot collected data for the desk scene while facing a different part of the room. This resulted in a "high conflict" type of state failure during fusion. The robot then generated a hypothesis of sensor malfunction, and attempted to run diagnostics on the two conflicting sensors. These diagnostics, however, showed that both sensors were working correctly, and at this point, the robot could not proceed further, and signalled for assistance.

At this point, the robot posts a request to the interactive exception handling panel of the blackboard, indicating the type of failure it has encountered, and including the beliefs which led to this failed fusion. In the initial version of teleVIA, the images leading to this conflict are also transmitted. The first knowledge source of teleVIA is activated with the precondition that a video camera is involved in the failure. This causes the video image to be displayed first, together with a list of preliminary hypotheses of what could be the problem. Examples of hypotheses include: wrong input, sensor malfunction, sensor occlusion, sensor hardware error (missing data, self-diagnostic error), multiple sensor errors and electromagnetic interference. Since the purpose of the system is to provide assistance as quickly as possible, an assumption is made that, if applicable, images which are most easily perceived by humans (e.g., video images versus thermal images) are given priority. Thus, if by looking at the video image, the operator can immediately ascertain what the problem is and resolve it, then this most effective solution to the problem should be supported. Once the operator selects the probable diagnosis, a list of repair possibilities may be posted to the interactive configuration panel for implementation.

Scenario 2

In the second experiment, the lens of the camera was covered in opaque plastic to simulate a sensor malfunction due to external factors such as dirt on the lens, for example. In this case, the fusion process resulted in a "below minimum" type of failure, and, as a result, the exception handling mechanism generated a first hypothesis of "inadequate sensing plan". A backup plan was then implemented, and sensor data was reaquired accordingly. The new plan added a color camera to the sensing suite, and subsequently a fusion failure of "high conflict" was encountered between the black and white camera and the color camera. As in the previous experiment, teleSFX then generated the hypothesis of sensor malfunction, performed diagnostics which denied this hypothesis, and then called for assistance.

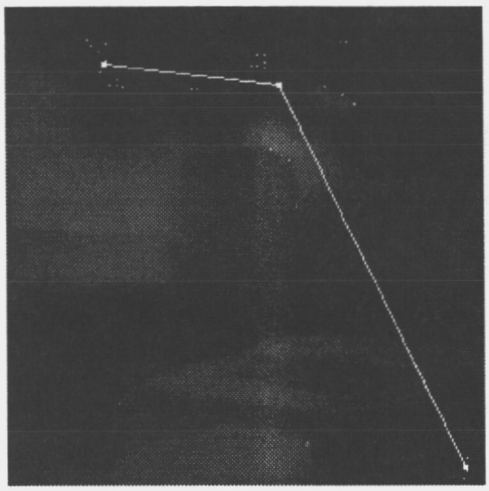


Figure 5: Example Image for Scenario 2.

In this case, both of the failures are posted to the blackboard, together with the beliefs generated for each attempt. Once again, the primary troubled sensor is the black and white camera, and the image generated is shown in Figure 5. In this case, however, since the second attempt introduced the conflicting image from the color camera, both the black and white, and the color video images are displayed by teleVIA for the operator to examine first. In this case as well, the operator should be able to determine the problem fairly quickly by simply comparing the black and white video image with that of the color camera.

Scenario 3

In the third experiment, the lights were turned off during data collection to simulate an unforeseen change in environment. In this case the exception handling mechanism of the robot arrived at a correct conclusion of "environmental change" by testing the visible light information. However, for this type of problem, operator assistance is still needed for recovery, and therefore a message is posted to the interactive configuration portion of the blackboard requesting intervention. The beliefs leading to the original state failure, together with the hypotheses generated and tested by the robot, are posted to the blackboard, while images and data from the relevant sensors (black and white camera, and UV sensor) are also displayed. This enables the operator to determine what type of environmental change may have occurred.

In each of these experiments, the primary sensor involved in the problem was the black and white camera. Since these experiments were originally designed to test the autonomous exception handling capabilities of the teleSFX system, the results, when extended to the teleVIA component are somewhat artificial. However, they serve the purpose to establish the type of

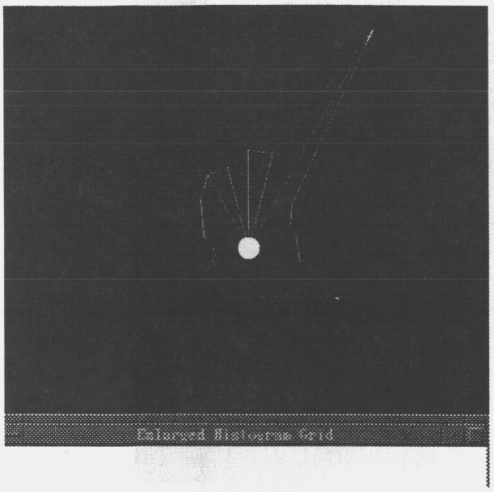


Figure 6: Polar Plot of Ultrasonics Data.

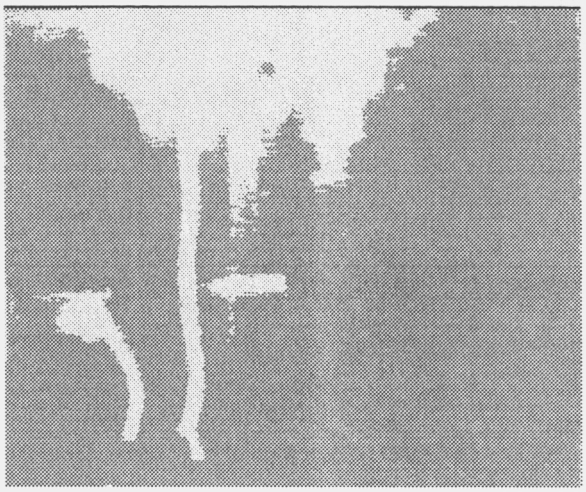


Figure 7: Equal Band False Color Enhancement.

information which must be communicated between the remote and the local systems in even such elementary scenarios. This allows us to determine the types of knowledge sources which may be activated, the different types of hypotheses which may be needed, and how to present this information effectively using the blackboard mechanism.

Further experiments are underway which emphasize sensor data which is not as easily perceived by the human operator, and which may require enhancement before conclusions may be drawn. In these cases, teleVIA knowledge sources are activated according to the type of sensor(s) involved in the state failure. This is then combined with knowledge of the current context to select appropriate enhancements, and display the information to the local operator. The following scenarios were constructed using images acquired by the robot for a drill press scene.

Scenario 4

In this example, it is assumed that the ultrasonics are contributing primarily to the fusion failure. In this experiment, one out of the 24 ultrasonics transducers mounted in a ring began to report widely fluctuating readings. A sensing failure of "highly uncertain" evidence was reported, but the responsible sensor could not be isolated, thereby necessitating aid from the operator. The raw ultrasonic readings that come from a Denning mobile robot are just numbers, which represent measurements in feet. However, when this data is represented as a polar plot as in Figure 6, it is much easier to notice if one or more of the sensors is giving erroneous readings. This is further reinforced if the numerical data are examined in the light of knowledge about the current context, for example, that a room (or mine shaft) is thought to have certain dimensions. A further enhancement of the data which can aid the local operator is an occupancy grid, which presents a bird's eye view of what the robot has sensed so far. The robot builds up this grid or map as it processes ultrasonics data. With both of these types of displays, the operator is more likely to diagnose the failure of an ultrasonic transducer or board, or to detect an erroneous reading.

Scenario 5

When the sensor in question during exception handling is the infra-red camera, enhancements are once again needed to assist the operator's perception of the information in the image. In this case, the untouched true infra-red image is typically gray scale, and there is often not a great deal of discernable contrast in the image. It is common practise to add false color to such an image to show the heat distribution. However, certain choices of false color maps still do not enhance the image, and may obscure the details even further. In the drill-press example, dividing the grayscale into 6 equal bands of color leads to a primarily yellow image, due to the extreme heat of the drill press. A grey scale version of this image is shown in Figure 7. However, if the selection of false color map is knowledge-based, utilizing model-specific information about the drill press, for example, then a more appropriately enhanced image is produced, making it easier for the operator to see the heat profile represented as blue, green, red, yellow, and white bands. This is illustrated in the grey scale rendition in Figure 8.

Future Work

Current work is concentrating on constructing experiments in real-time where an operator at Clark Atlanta will interact with the remote robot (Clementine) at Colorado School of Mines. An important issue which has not been addressed in this work so far is that of learning. The robot will typically be working in hazardous and/or remote environments about which little may be known, and therefore it is difficult to anticipate the types of problems which may arise. Not only would it be desirable to increase the autonomy of the individual robot wherever possible, but the knowledge gained from solving these problems could be disseminated to other robots in the field. Further-

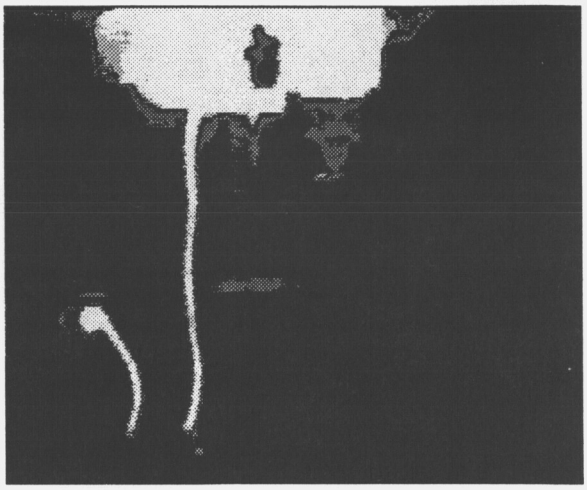


Figure 8: Model-Specific False Color Enhancement.

more, if the teleVIA system could "remember" certain interactions, these could immediately be retrieved from memory, rather than having to generate the same session over again. The technique of case-based reasoning is a natural candidate for this type of learning. Each interactive exception handling session may be captured as a case, which would be indexed on features such as particular configurations of sensors and failure types. Such a case could also include relevant images, or at least image types and enhancements used, so that teleVIA would simply use a case retrieval mechanism rather than a potentially complicated reasoning strategy. Certain aspects of the exception handling and recovery procedures might also then be communicated to the robot itself, to extend its autonomous capabilities, especially for recurrent problems.

Summary and Conclusions

This paper presents a new approach in semi-autonomous mobile robots, which reduces the level of human supervision and provides intelligent assistance for problem solving. The approach partitions problem solving responsibilities between the remote and the local machines. The remote system monitors its sensing for anomalies, called sensing failures, using teleSFX. If a failure occurs, it attempts to classify the source of the problem using a generate and test methodology. If it is successful in identifying the source, it then attempts to autonomously recover (e.g., go to back up sensors, change parameters). Otherwise, if the source cannot be classified, or if no recovery strategy is available, the local machine must provide the exception handling. Exception handling at the local is done by the operator, with the help of teleVIA. TeleVIA uses a common blackboard to cooperatively assist the operator by posting what has been done by the remote, displaying and enhancing sensor data needed in ascertaining the problem, and managing diagnostic hypotheses and beliefs. Experimental scenarios using data collected from mobile robots illustrates the operation of the system.

The advantages of this type of tele-assistance fall into three categories. First, it is practical. It reduces the need for direct human supervision and communications by having the remote monitor itself for failures. Second, it increases efficiency by freeing the operator to supervise multiple remotes, reducing cognitive overload by controlling the presentation and enhancement of sensory data from the remote, and aiding the problem-solving process through hypothesis management and expert guidance. Three, the approach supports the incremental addition of artificial intelligence as more progress is made in learning and planning.

References

[1] Chavez, G. T., and Murphy, R.R., 1993. "Exception Handling for Sensor Fusion" SPIE Sensor Fusion VI, Boston, MA, Sept. 7-10, 1993, pp. 142-153.

[2] Coiffet, P. and Gravez, P., 1991. "Man-Robot Cooperation: Towards an Advanced Teleoperation Mode". In S. G. Tzafestas (Ed.), *Intelligent Robotic Systems.* Marcel Dekker: New York, pp. 593-636.

[3] Murphy, R.R., 1993. "Robust Sensor Fusion for Teleoperations", 1993 IEEE International Conference on Robotics and Automation, invited special session on Multisensor Fusion, Atlanta, GA, May 2-6, 1993, vol. 2, pp. 572–577.

[4] Murphy, R.R. 1992. *An Architecture for Intelligent Robotic Sensor Fusion.* Technical Report No. GIT-ICS-92/42, College of Computing, Georgia Institute of Technology, Atlanta GA 30332.

[5] Rogers, E. 1992. *Visual Interaction: A Link Between Perception and Problem-Solving.* Technical Report No. GIT-CC-92/59, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332.

AIAA-94-1246-CP

MOBILE ROBOT EXPLORATION AND NAVIGATION OF INDOOR SPACES USING SONAR AND VISION

David Kortenkamp*, Marcus Huber, Frank Koss, William Belding,
Jaeho Lee, Annie Wu, Clint Bidlack and Seth Rodgers

Artificial Intelligence Laboratory
The University of Michigan
Ann Arbor, MI 48109

Abstract

Autonomous mobile robots need to integrate many different skills in order to perform complex tasks. In particular, they need to explore, sense, map and navigate in unknown or partially known environments. This paper describes a robot system that is designed to perform a find-and-deliver task in an office-building-like environment. The robot's initial orientation and location within the environment are not known, but the robot does have an *a-priori* map of the environment. We describe a sensor-based map representation that the robot uses while exploring its environment. We also describe how the robot determines its initial position and orientation within the environment, how it explores the environment for a visually-tagged object, how it recognizes the object and how it delivers the object. The robot also updates its map to reflect changes in the environment. While the entire robot system has not yet been integrated, each subsystem described in this paper has been implemented and tested.

Introduction

Autonomous mobile robots need to explore, sense, map, navigate and perform tasks in the environments in which they find themselves. Often these five functions are studied separately, with little or no attention given to how they are all integrated to produce a completely autonomous mobile robot. In this paper we concentrate not on completely describing any single aspect of robot exploration, sensing, mapping or navigation, but instead on how many different skills can be integrated into an autonomous robot that performs a sophisticated task. Unfortunately, time constraints prevented a complete integration of all of the described skills on the mobile robot. All of them, however, were tested individually and their integration is planned.

*Now at The MITRE Corporation, 1120 NASA Road 1, Houston TX 77058, e-mail: korten@aio.jsc.nasa.gov

Copyright © American Institute of Aeronautics and Astronautics, Inc., 1993. All rights reserved.

Task description

The task our robot is designed to perform is to find a single, visually tagged object somewhere in a large, office-like environment and to "deliver" the object to a designated room. The robot is given a crude map shortly before being asked to perform the task. However, the map does not show obstacles that may block hallways or doors, nor does the map show all of the doors in the environment. The robot does not know its starting position or orientation with respect to the map. The delivery object is in one of the rooms and is a coffee pot marked with a black-and-white 'X'. The robot need not actually pick-up the coffee pot, only approach it. Some, but not all, of the doors are tagged with a visually distinct bar-code; bar-coded doors are noted on the map and the delivery room will be one of them. The robot has 30 minutes to complete the task, which was one of three tasks that comprised the AAAI '93 Robot Competition and Exhibition held in Washington DC on July 11-16, 1993.

The task is challenging to mobile robots because it requires the integration of many mobile robot skills. The robot must initially explore the environment and determine its position and orientation with respect to the *a-priori* map. The robot must then plan an exploration strategy that will allow it to examine each room for the coffee pot. This strategy must be flexible in the face of unexpected obstacles. Finally, the robot must use visual sensing to detect the coffee pot, plan a path from the object to the delivery room and then follow that path.

Robot description

Our robot is a Cybermotion K2A called CARMEL (Computer-Aided Robotics for Maintenance, Emergency and Life Support) (see Figure 1). It has a ring of 24 sonar sensors and a rotating B&W camera. Three computers are on-board CARMEL, one computer each for the motors and sonar sensors and a 486-PC for high-level processing. The 486-PC has a framegrabber and performs all image processing. CARMEL has a basic obstacle avoidance competence

Figure 1: The mobile robot CARMEL.

provided by an algorithm called VFH [2, 3, 4]. VFH constructs a certainty grid of sonar hits and uses it to continually compute a new direction that will take the robot towards its target while avoiding obstacles.

Overview

We first present the robot's representation of its environment. This is the representation that is entered into the robot from an *a priori* map. The robot must then *register* itself (i.e., determine its orientation) with respect to the environment; we give a basic registration algorithm. Next the robot must *localize* itself with respect to the *a priori* map; two different localization algorithms are presented. Once the robot is registered and localized, it can begin exploring the environment and looking for the coffee pot. We describe our vision algorithm to detect the coffee pot and also describe how we update the *a priori* map to reflect changes in the environment. Finally, the robot must navigate from the room that contains the coffee pot to the delivery room. This sequence is shown in the flow chart in Figure 2.

Representing the environment

The map in our representation is a graph of nodes. Each node represents a region of the environment that has a common sonar signature. Each link between nodes represents a bidirectional connection between the two regions. The first issue when creating such a representation is to decide on an appropriate sonar signature that will distinguish between different regions.

Detecting region boundaries

There have been many approaches to using sonar sensors to define distinctive places in an environment, including [10, 1, 6, 7, 9, 8]. In our approach, a region in the environment is characterized by having a common sonar signature throughout its extent, where the

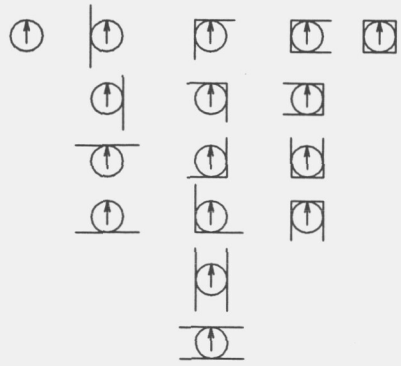


Figure 3: The sonar signature feature set detectable by CARMEL.

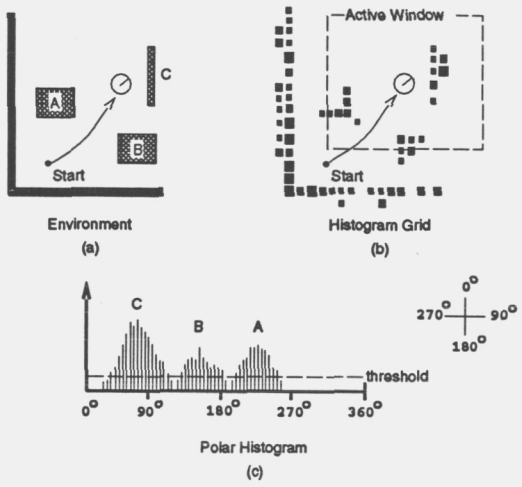


Figure 4: The VFH obstacle avoidance algorithm.

signature is the pattern of free or blocked space to the front, back and sides of the robot. Thus, there are 16 unique sonar signatures in a rectilinear environment (see Figure 3 for a complete listing of the 16 sonar signatures). Our approach is unique in that it is directly tied to an obstacle avoidance algorithm—the Vector Field Histogram (VFH) [4].

The VFH algorithm first creates a histogram grid, which is a certainty grid representation of the objects surrounding the robot as detected using the robot's sonar sensors. VFH then takes a local window of the certainty grid and converts it into a polar representation called the polar histogram. A certainty grid and its corresponding polar histogram are shown in Figure 4. The polar histogram shows the obstacles in each direction around the robot. To avoid obstacles, VFH simply chooses the free direction of travel that is nearest to the desired direction of travel. This same polar representation is used to produce the sonar signature.

A simple example will best show how the polar histogram is used to detect a region boundary. The robot is started down the hallway (the direction of the hallway is determined by an algorithm described in Section 3) and VFH automatically aligns the robot and

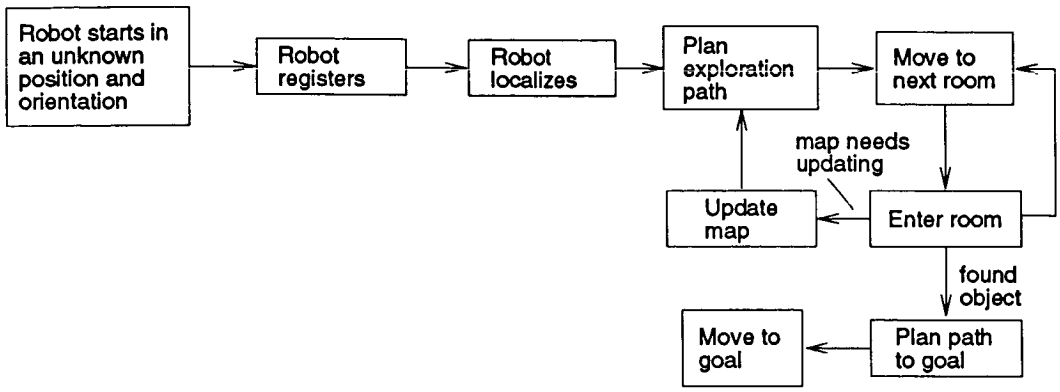


Figure 2: Flowchart for accomplishing the find and deliver task.

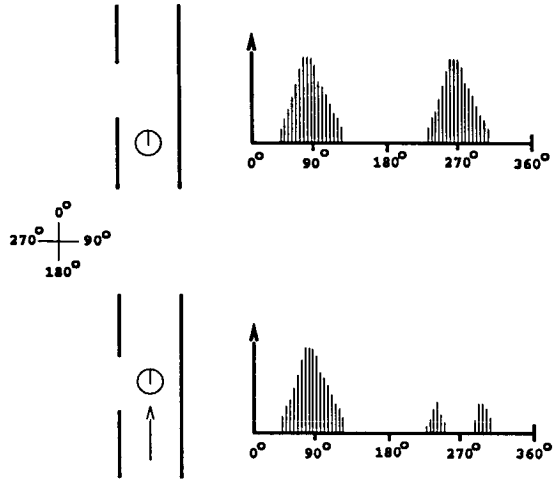


Figure 5: Detecting region boundaries using VFH.

positions it in the middle of the hallway. When the robot is positioned in the middle of a hallway the polar histogram has two "mountains" for the two walls of the hallway (Figure 5(top)). The presence of a "mountain" means that the robot is blocked to that side. In this example, the sonar signature is: (front = open, back = open, right = closed, left = closed). As the robot moves down the hallway and approaches the doorway, the "mountain" on that side of the robot will disappear (Figure 5(bottom)). So the sonar signature is now: (front = open, back = open, right = open, left = close). By "camping out" at the polar histogram segments corresponding to the front, back, left and right of the robot, changes in the sonar signature can be immediately detected.

In tests on the repeatability of this algorithm, CARMEL was asked to repeatedly stop at the same region boundary in the basement of our laboratory. Over ten consecutive runs, the largest difference in position along the hallway's axis between any two runs was 520mm and the largest difference in position perpendicular to the hallway axis along any two runs was 290mm. During these runs, obstacle avoidance was performed and the robot was running at a speed approaching 400 mm/sec.

While our boundary detection algorithm works fine in hallway environments, it has not been extensively tested in rooms. We rely instead on the dead reckoning capabilities of our robot to move into and out of rooms. Extending our approach to rooms as well as hallways is a topic of future research.

Map representation

Each region of the environment, which corresponds to a sonar signature, is represented by a node. A node contains the extent of the region (i.e., its length and width), a global (x,y) position of the center of the region and connections to neighboring regions. Figure 6 shows an example configuration of the arena where the robot will be working. As shown in Figure 7, the whole area is divided into regions based on the sonar signature. The regions are further distinguished by either being a hallway region or a room region. Each hall section has one node at the center of the hall (nodes with "H" prefix). Every exit of the room also has one node close enough to the entrance (nodes with "R" prefix). Each room section has some extra virtual nodes (nodes with "V" prefix) for each side of the walls of the room. These virtual nodes serve two purpose. First, they are used to figure out the boundary of the room. Since each node has its (x, y) coordinate, we need at least two room-nodes to calculate the boundary of a rectangular shape room. Second, they are used for map modifications which will be explained later in this paper.

Rooms can have up to four exits, one each to the north, south, east and west. If a room has more than one exit on each side it will be split into several virtual rooms. Large open space, such as lobbies, are also classified as rooms and may have to be split into several virtual rooms. For example, rooms 7,8 and 9 in Figure 7 are all virtual rooms contained within a single open area.

Registration

In order for our representation scheme to work, CARMEL must be able to determine the main axes of the corridors, so that it can start searching for region boundaries to its left, right, front and back. We call this *registration*. Currently, CARMEL can only

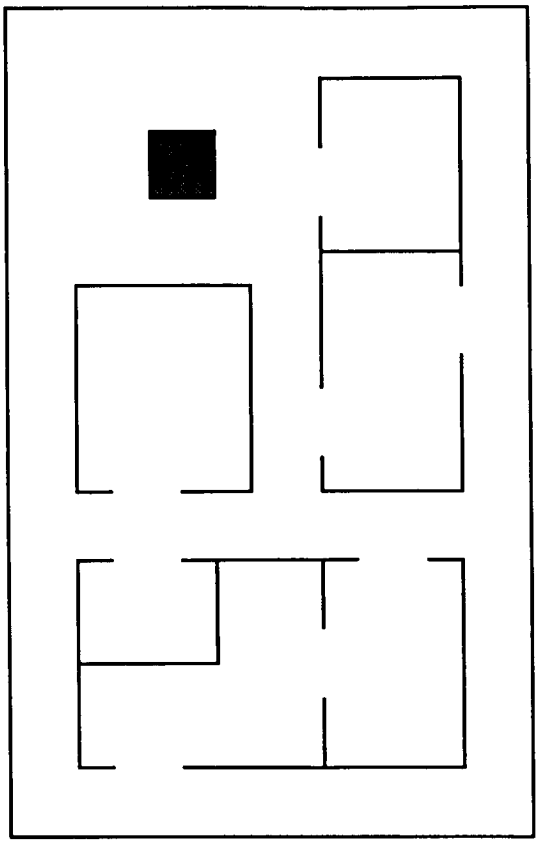


Figure 6: Arena Configuration

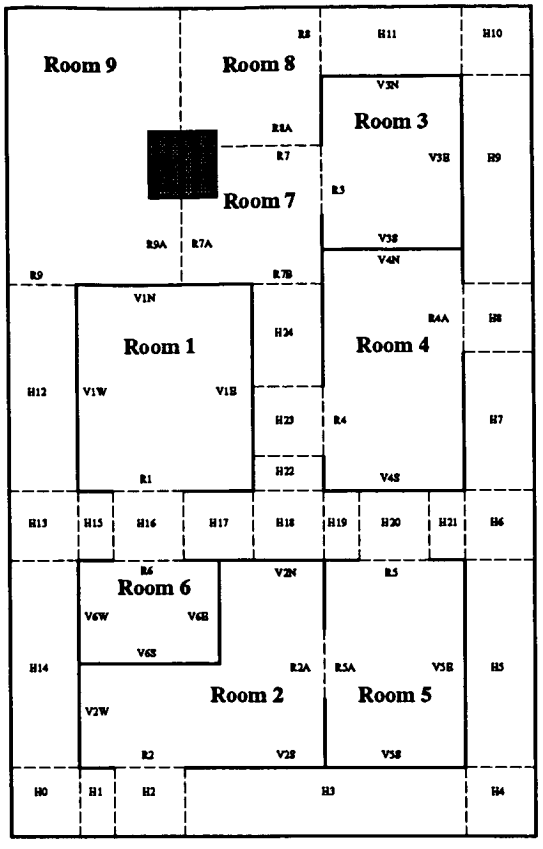


Figure 7: Arena Map

register itself in a hallway; if CARMEL starts in a room it must wall-follow until it enters a hallway. To register in a hallway, CARMEL starts to travel in any free direction. As it travels, the VFH obstacle avoidance algorithm will automatically align CARMEL between the two walls of the hallway. While moving, CARMEL saves its (x,y) positions along the way and fits a line to them. The orientation of this line is used to determine the axis of the hallway. CARMEL can reregister during the task to correct dead reckoning errors.

During initial registration, when the robot has no information as to its orientation in the environment, CARMEL also stores and averages the direction of free space. This should always fall along the axis of the hall. However, obstacles in the hallway, doorways, and intersecting corridors cause CARMEL to drift from the middle of the hall. Therefore line fitting, which is a simple chi-square fit, is used. This occurs after CARMEL has traveled a minimum distance and the rate of change in the average free-space direction falls below a threshold. If CARMEL becomes trapped before this time, it turns around and starts the process again assuming that it has reached a blockade in the passage or the end of a hall. If the orientation of the fit line is too far from the average free-space direction, it is assumed that CARMEL has not been traversing a hall or has "fallen" into a room or an intersecting hallway. In either case, all data are disregarded and the entire process, including wall-following if necessary, is repeated.

For reregistration during the task, the previous orientation can be used to judge the accuracy of the calculated orientation. When there is a large difference between the previous and new orientations, either the old one can be maintained or the process can be repeated. Maintaining the old orientation repeatedly is dangerous because CARMEL's orientation can become very inaccurate over a period of time.

We evaluated the registration algorithm in two situations. The first situation was in a corridor with no obstacles or openings into rooms or intersecting hallways. These experiments set out to confirm that the algorithm will correctly identify the hall axis regardless of CARMEL's initial orientation. In the second set of experiments, CARMEL was placed in a more complex area which included an obstacle and an opening into a room. The intention of these runs was to determine the robustness the algorithm, as it currently stands, in a more realistic situation.

Twenty-seven runs were carried out in the obstacle-free hallway. CARMEL's initial orientation with respect to the hall axis varied from 10 to 170 degrees in 20 degree steps. CARMEL's initial orientation was determined by eye and so is inaccurate by up to a degree or two. At each initial orientation, three runs were made. CARMEL determined the actual hallway axis to within six degrees in all but one run. In this case, the calculated hall axis was off by nine degrees. No run required more than approximately four

meters. The accuracy of the registration and the distance covered during a run were acceptable and within the limits of the environment that was expected to be encountered.

The second set of runs had two parts. The first part was carried out with CARMEL given an initial orientation of 30 degrees. The second part used an initial orientation of 70 degrees. Eight runs were carried out with each orientation. In the first part, CARMEL determined the hall axis to within five degrees each time, except one in which it wandered into the room through the opening. The average distance required was approximately five meters. In the second part, CARMEL entered the room twice, but determined the hallway axis to within four degrees in the other six runs. The average length of the successful runs was about 2.6 meters.

While these runs were far from exhaustive, they do show that this method of registration is useful. The most difficult problem is that of wandering into a room. This can either be avoided or detected, with the first preferable. The difficulty with preventing CARMEL from drifting into a room is that there is no simple way to distinguish between an opening into a room and a narrowing of the corridor due to obstacles. The former should not be entered while the latter should be. Detecting the entry into a room should be simpler. The chi-square fit provides a goodness of fit measure, namely χ^2. When CARMEL enters a room, its path is generally straight but roughly perpendicular to the hall axis. This should yield a very poor value for χ^2. The use of this value and the return of CARMEL to the hallway it left are still being investigated.

Localization

Once registered, the next critical issue is the determination of the correct location and orientation of the robot. We call this process *localization*. CARMEL accomplishes localization through the accumulation of information, in the form of local sonar signature features, during its initial movement through the halls of the "office" environment, and through observation of visual tags identifying doors. We would like CARMEL to localize itself as quickly as possible, however, so that in the absence of door markers we try to use the sonar signature features. In both approaches CARMEL is given a map representing the environment in which it will be placed. However, the map can be in error in that doorways may exist where they are not so indicated on the map, and doorways may be blocked where they are indicated on the map. The localization schemes must therefore deal with these problems.

We have implemented two approaches, one based upon heuristics and confidence factors, the other upon probabilistic reasoning using a belief network. Our localization methods only work in hallways, so that if CARMEL's initial location was within a room we would first have to find an exit using a wall following behavior. In this section we describe each of the approaches and show them in operation. Although both were implemented, neither have actually been fully integrated into the office exploration system.

Rule Based Localization

One method of localization that has shown to be successful is a rule based system. Before CARMEL makes a move during rule based localization, it computes scores over all of its possible starting locations in the *a-priori* map.

Creating a Score Distribution

Since direction is ambiguous to CARMEL at first, we run our scoring algorithm four times, rotating CARMEL's map to a new cardinal orientation each time. So for n possible starting locations, there are $4n$ total scores computed.

The basic scoring algorithm is a modified depth-first recursion, which runs as follows:

The first feature node seen by CARMEL is compared with the start node in this orientation. The comparison scores points depending on how many sonar signature features (i.e., walls or openings on the four sides of the robot) match and on the measured extent of a region. However, points scored for extent matching are fewer because we have determined that distance data tend to be more erroneous than the detected sonar signature features.

Each node adjacent to the current one on CARMEL's constructed map is checked to see if it has been examined yet in this orientation. If the neighboring node has not yet been examined, then it is compared with the node corresponding to it on the *a-priori* map. The algorithm continues this recursively for all of the paths the robot can follow from the start node. The score for each recursion is added to the total score for that start node in that particular orientation.

While the algorithm recurses, it also tries to answer this question:

> *If CARMEL started in this start node with this orientation, where would CARMEL be now?*

If the algorithm can determine this, it makes note of the fact. We refer to a current location inference of this type as a *location resolution*. There is no guarantee that a start node–orientation configuration will produce a location resolution.

In the event that a path to an adjacent node exists in CARMEL's map but a wall exists on the *a-priori* map, the routine attempts to figure out possible locations across the wall that might correspond to the node CARMEL saw. If a possible match is found, then the routine continues from there; otherwise, no points are scored for that area on CARMEL's map.

After all of the location–direction combinations are examined, the algorithm normalizes the raw scores by computing the mean and standard deviation of the score set. Then each original score is replaced by the number of standard deviations the score was above the mean. The resulting scores are now less dependent on the number of nodes seen by CARMEL.

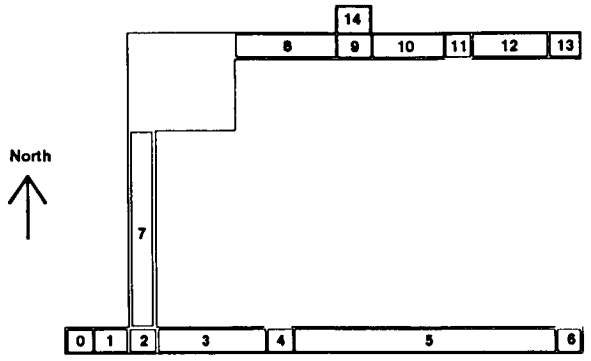


Figure 8: Map of the experimental space showing a priori regions.

Move Planning

CARMEL scores the possible moves it can make based on the location resolutions. CARMEL looks at each location resolution that it has and computes a shortest path to the nearest door marker for that resolution. The first move of this path is considered. This first move is either one of the four directions, or no movement at all (the special case where CARMEL is hypothetically in the vicinity of a door marker).

For each first move of a particular type, weight is added to that corresponding move possibility. The weight added depends on the score found for the location–orientation pair that the first move was derived from. If CARMEL saw a door marker at this stop, or if any location–orientation score for one of the first moves is above a certain threshold, then the "don't move" move choice is given a score of infinity, and CARMEL assumes localization is complete. In the former case, CARMEL assumes it is now at the location on the *a-priori* map where the door marker is. In the latter case, CARMEL assumes it is at the location resolution found for the above–threshold score.

As a final factor in move choice, we programmed CARMEL to select from these possible movement choices the highest scoring direction that has the shortest path to an unexplored region on CARMEL's map. This ensures that CARMEL covers unexplored space as efficiently as possible while searching out door tags. It also guarantees that CARMEL will not 'paint itself into a corner' or oscillate between adjacent nodes while exploring (two problems that occurred without this adjustment).

Experimental Results

Here are some results of a typical run with the rule based algorithm. The map given CARMEL is shown in Figure 8, and a door tags are located at nodes 0, 6, and the north end of 7.

We placed CARMEL in feature region 2 and aligned the robot so that it faced south on the map. CARMEL was told that it was starting somewhere along the south hall (nodes 0–6). CARMEL localized after the third move without using door markers.

In Table 1, the Move # column shows the current move. The Best Resolved column reports the best location–direction pair, i.e., CARMEL's top choice(s) for where it may be. The number is the feature node label corresponding to the map, and the direction is the direction CARMEL thinks it's facing. For example, 2–south means CARMEL thinks it may be at node 2 facing south.

The 'Best' Move(s) column indicates the best moves computed by the possible move scoring routine. Directions are displayed here as the true direction on the map for ease in interpretation. Multiple directions indicate a 'tie', in which case the final move is chosen from them based on exploration preference.

The Move Choice column is the actual move made by CARMEL. It may differ from the previous column if CARMEL chose an unexplored area over a high score direction.

It may seem strange at first that the 'best' move and the move choice are totally uncorrelated after the first move. One must remember that the 'best' move is a result of weighting all possible moves for all resolvable pairs, so it doesn't necessarily represent the true best move that can be made, especially in a symmetric environment. The moves chosen were carried out because CARMEL picked a direction, and preferred to explore new area on a 'next best' score rather than backtrack on a best one (which only may be best by a margin). Also, it is important to remember that the Best Resolved nodes are not the only nodes used in determining direction. All of the possible location resolved nodes are considered, weighted only by the location–orientation score associated with them. The Best Resolved values are therefore only displayed to show how quickly the algorithm can localize.

Belief Network Approach

In the second localization approach, the dependence of the sensed features on the world map, the robot's initial orientation, and the direction of travel of the robot as it attempts to localize itself, is modeled using a belief network [5, 11]. As the robot moves about and sees new features, the belief network accumulates a history of the features observed and the movements that the robot has made. These observations can then be propagated through the network, resulting in a probabilistic distribution over the possible locations the robot may be in currently. The robot considers itself localized when one of the locations achieves a level of confidence about a certain threshold. If CARMEL is not yet localized, it can use this distribution to determine the most likely direction in which to travel to facilitate better localization. Currently, this amounts to moving in the direction most likely to take it to a room tag, the most unambiguous localization feature detectable by CARMEL.

Belief network operation

The belief network that we used is shown in Figure 9. This network models the dependencies between the robots initial location, its initial orientation, and the sonar feature that it "sees". The modeling is accomplished both through the topology of the network as well as the probability tables (both conditional and priors). The conditional probability that a certain

Move #	Best Resolved	'Best' Move(s)	Move Choice
start	2-south,4-south	east,west	east
1	3-south,5-south	west	east
2	4-south	west	east
3	5-south	no move	no move

Table 1: Results from an experiment using rule-based localization.

feature is detected, given a particular location and orientation, is based on heuristic calculations of the correlation between what should be observed and that which is actually observed. The conditional probability that the robot is in a particular location, given a previous location and orientation, is a simple boolean function based on the map, where the probability is 1.0 if the locations are adjacent and joined by a path, and 0.0 if they are not.

Upon initialization, an observation of the robot's initial surroundings is placed in the FEATURE1 node of the network and then propagated throughout the network. The resulting posterior probability distribution in the LOCATION1 and ORIENTATION nodes reflect the evidence's impact upon the likely starting location and orientation of CARMEL. The robot can use this revised information in its planning to either facilitate improved localization or to switch to exploration of the office environment if the probabilities are suitably high enough to justify this.

In the situation where the resulting probabilities are such that CARMEL is still unsure enough of where it is to warrant further localization, CARMEL plans and executes a move in a direction most likely to take it to a door tag in the shortest amount of time (i.e., shortest distance). When it detects a change in the sonar features around it, it stops and makes another observation. The new sonar feature, as well as the motion the robot made to get to its current location, is fed to the belief network as evidence, propagated, and the resulting probability distributions analyzed. This cycle continues until either CARMEL becomes sure enough of its location based solely upon the sonar features so far detected, at which time it switches to exploration mode, or CARMEL detects a door tag, at which time it knows with certainty where it is, and similarly switches to exploration mode. The belief network in Figure 10 shows the belief network at iteration 3 in the process. As can be seen in this figure, the belief network grows at each iteration, adding new LOCATION, MOTION, and FEATURE nodes. The portion of the network that includes the MOVE node models the dependence of the robot's new location upon the previous location, the original orientation, and the move the robot made to get to the new location.

Experimental Results

We evaluated the belief network's ability to localize CARMEL in the halls of the University of Michigan's Artificial Intelligence Laboratory. The map for the region is shown in Figure 8, with the possible localization locations indicated by the numbered regions. Each of the labeled locations represents a region of

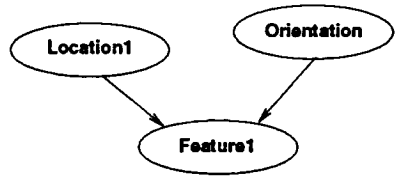


Figure 9: Initial belief network architecture.

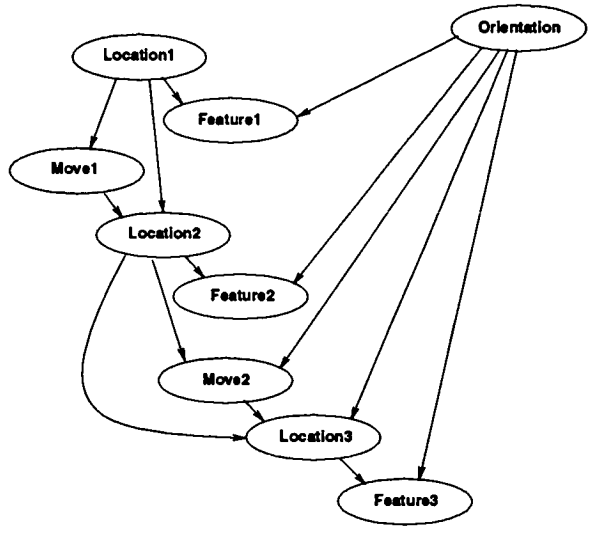


Figure 10: Belief network architecture showing the network at iteration 3 of the process. The Feature and Move nodes are instantiated as evidence, and the Location and Orientation nodes are inferred.

the map that has the same sonar feature type. Travelling between regions (locations), then, implies that the sonar feature must change at the transition point between the regions.

As an example, suppose CARMEL starts in location 2, the T-intersection at the South end of the map, and is initially facing South. The sonar feature observed would be that of a single blocked direction, that directly in front of it. Passing this evidence to the localization network, the resulting probability distribution for the current location is shown in Table 2(top), while the posterior distribution of the ORIENTATION node is shown in Table 2(bottom). These state that CARMEL is either in Location 2, 4, 9 or 11, and is most likely to be facing South.

If CARMEL then moves West (to it, East on the map), it will move until it enters region 3, which has a different sonar feature. The new feature, that of an East-West hall, and the West move that CARMEL made, are both given to the localization network and

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0.04	0.04	0.13	0.04	0.13	0.04	0.04	0.04	0.04	0.13	0.04	0.13	0.04	0.04	0.04

North	South	East	West
0.27	0.44	0.15	0.14

Table 2: Probability distribution of location (top) and orientation (bottom) at the start location.

propagated. The new probabilities for the current location and orientation are show in Table 3, which says that CARMEL thinks that it most likely to be at Location 10, and is most likely to be facing South. Again moving West (to it, East on the map), so that CARMEL sees the new feature at Location 4, yields distributions shown in Table 4.

Taking this probability distribution, CARMEL now has greater than 90% confidence that it is at Location 4 and was initially facing South. If there was a door tag at the entrance to the room at Location 4, it could then visually verify that this inference is correct. CARMEL can now reorient itself correctly to the map and position itself at the transition point between the current location (Location region 4) and the previous location (Location region 3). Note that since the belief network has nodes representing each of the previous locations also, it is easy to reason about where the robot has already been simply by looking each of the nodes and determining the highest probability state in each. This facilitates exploration updating in that extra time to perform a backtracking search through the map with the previously performed motions doesn't have to be done in order to see what has already been explored.

Exploration and Navigation

Once CARMEL is localized it can begin to look for the coffee pot. The first step in this process is to plan an exploration path. An exploration path is an exhaustive sequence of rooms to visit from the robot's current location. The sequence is determined based on the travel distance from the current location. CARMEL first selects the closest room (in terms of travel distance, not the Cartesian distance), then adds another room closest to the selected room, and so on. After planning, CARMEL traverses the exploration path stopping in each room to scan with its camera for the coffee pot. Exploration is terminated when the coffee pot is found. The exploration path can be modified to accommodate unexpected blockages or openings.

Planning exploration path using closest-node-first (hill-climbing) method does not necessarily generate the optimal path in terms of total traveled distance*, but in practice this method turned out very fast and the resultant path was quite reasonable. For example, the exploration path from, say, "H18" (in the middle of the map in Figure 7) would be (R4, R7B, R3, R8A, R9A, R1, R6, R5, R2A). This sequence specifies only the room nodes to visit in that order.

*Finding the optimal exploration path amounts traveling sales man problem

Updating the map

While exploring CARMEL can update its *a-priori* map to reflect blocked hallways and doorways and to note additional doorways that were not in the original map. For blocked hallways or doorways, the connections between the two nodes in the map are cut and the sonar characteristic of the node is modified appropriately. In the case of unexpected openings, new nodes are created, assigned the appropriate signature and connected to adjacent nodes. This information helps CARMEL find the most efficient route to the delivery room once the coffee pot has been found or to replan its exploration path. For example, if the robot sees an unexpected opening to a room from a hall node, it finds the nearest node (either virtual or real room node) of the room and create a new link to that node. Note that each node can have maximum four connections (roughly corresponding to North, West, South, and East) to other nodes. Figure 11 shows a part of the map before the robot sees an unexpected opening at west side of the hall "H5". It first figures out which room is next to west of "H5" from the boundary information of each room. In this case, it is Room 5 and "V5E" is the closest node of that room. Now "H5" should be divided into three sections since the sections are divided based on the sonar-reading changes and new opening will change the sonar signature as shown in Figure 12.

Visual sensing

As CARMEL enters each room it scans for the coffee pot. CARMEL's vision system finds predefined markers (a black 'X' on a white background in the case of the coffee pot) in the environment and determines their pose (3D position and orientation) relative to the robot. We will describe the algorithm for detecting the 'X' here. The algorithm for determining the pose of the 'X' is described in [12].

Marker detection

The marker detection phase is composed of two main routines: the connected components routine and the marker identification routine. The detection phase must be both fast and accurate for the pose estimation algorithm to be useful for real-time tasks.

To maximize speed, we make only one pass through the entire image. During the pass, the image is thresholded and connected components are found and labeled. One pixel components are ignored and not labeled. Size thresholding then filters out most of the non-marker components. Only one pass is made through all possible connected components.

To identify or reject the remaining markers, we use a weighted pattern matching template. An *nxn* template matrix is created for each marker (see Figure 13).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
< 0.01	0.04	0.01	0.20	0.01	0.20	0.01	0.03	0.05	0.01	0.36	0.01	0.04	< 0.01	< .01

North	South	East	West
0.33	0.62	0.04	0.02

Table 3: Probability distribution of location (top) and orientation (bottom) after first move.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0.01	< 0.01	0.08	< 0.01	0.57	< 0.01	0.08	0.01	0.02	0.07	< 0.01	0.13	< 0.01	0.01	< 0.01

North	South	East	West
0.18	0.800	0.01	0.01

Table 4: Probability distribution of location (top) and orientation (bottom) after two moves.

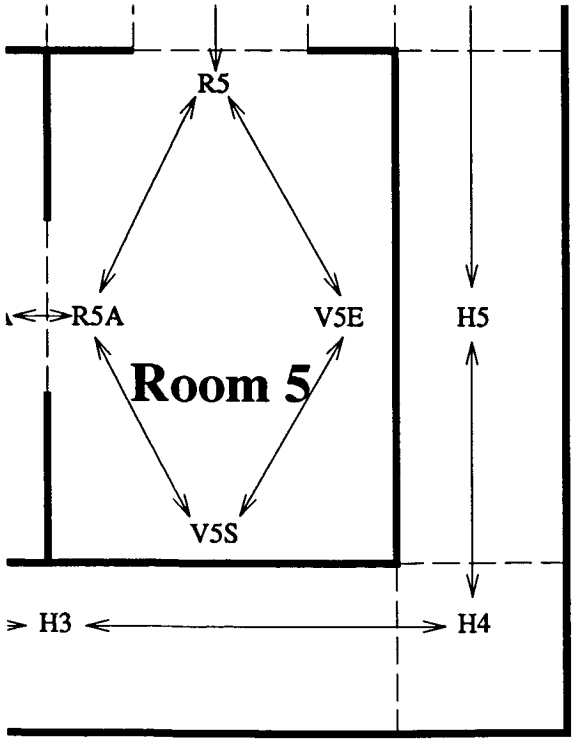


Figure 11: Before Modification

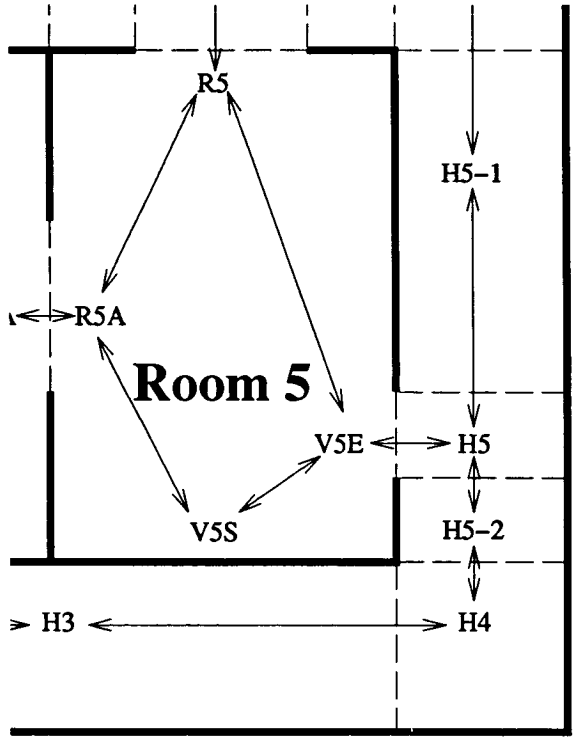


Figure 12: After Modification

1	1	-2	-8	-2	1	1
1	2	0	-1	0	2	1
-2	0	3	1	3	0	-2
-8	-1	1	8	1	-1	-8
-2	0	3	1	3	0	-2
1	2	0	-1	0	2	1
1	1	-2	-8	-2	1	1

Figure 13: Weighted pattern template for the 'X' markers. Positive values indicate expected black areas; negative areas are expected to be white. Certainty increases with magnitude.

b	w	w	w	w	b	b
b	b	w	w	b	b	w
w	b	b	b	b	w	w
w	w	b	b	b	w	w
w	w	b	b	b	b	w
w	b	b	w	w	b	b
b	b	w	w	w	w	b

Sample marker

$$Certainty_x = \frac{\sum_r \sum_c f_x(r,c)}{\sum_r \sum_c |x_{rc}|} = \frac{92}{96} = 0.9583$$

$$f_x(r,c) = \begin{cases} |x_{rc}| & \text{if correct color} \\ 0 & \text{otherwise} \end{cases}$$

Figure 14: Sample marker with calculated 'X' certainty value. "b" indicates a black pixel; "w" indicates a white pixel. r counts rows; c counts columns.

Increasing n increases the resolution of the template, but also increases the process time. We found $n = 7$ to be a good compromise. This weighted template indicates which areas are expected to be black and which ones white. The weights for our matrix are currently determined by trial and error, but we could easily replace these with machine generated weights if a learning program were implemented. The marker template which a component most resembles is selected as the "guess" for that component. The program generates a certainty measure with each guess (see Figure 14) and uses this measure to accept or reject the guess. Each marker can have one or more templates. The additional templates may be used to improve marker recognition from views other than straight on.

We also use additional heuristic information in identifying the markers. Some heuristics were not learned or incorporated until after the program had been tested. For example, diagonal lines often scored high enough certainty values to be considered 'X's. Once we realized this, adding a specific test to verify that each possible 'X' is not a diagonal line solved this problem. To avoid slowing down the program too much, specific heuristic tests were kept to a minimum.

Navigation

Once the coffee pot is found, CARMEL uses the vision algorithm's estimation of the pot's relative location to approach it. Since CARMEL doesn't have a manipulator, it is assumed that once CARMEL has approached the coffee pot it has "grabbed" it and can then deliver it to the delivery room. CARMEL plans the shortest path to the delivery room using a standard shortest-path algorithm. CARMEL then follows the path by moving from region to region and detecting region boundaries with its sonar sensors. There are numerous error recovery routines that can cope with changes in the environment and sensor errors.

Conclusion

Unfortunately, time constraints leading up to the competition prevented the complete integration of all of the described skills. In particular, the robot did not perform registration or localization at the competition. Instead the robot was told its orientation and position. During the actual competition, the robot explored several rooms before becoming hopelessly lost, at which time the run was terminated. The most difficult problem encountered was tuning the sonar-based region-finding algorithm to the particular environment. While the algorithm had worked fine in our testing environment (the basement of our laboratory), different characteristics of the competition environment caused many false detections (i.e., defining the start of a new region when there wasn't one) and a few missed detections (i.e., not detecting a new region when there was one). Since the robot's localization depended on matching the regions it found with the *a priori* map, it became lost very quickly.

Our experience demonstrates an important lesson in mobile robotics—if the low-level sensing of the world is not working correctly, then high-level reasoning or map making will be unsuccessful, no matter how elegant their implementations. Our experience also underscores the fact that routines that are demonstrated to work in one environment will not necessarily work in another environment, even if that environment is quite similar. In addition, our experience was not unique—no robot at the competition (out of a dozen entries) successfully completed the task. Obviously, there remains much work to be done in mobile robot exploration and navigation of indoor environments.

Acknowledgments

The authors wish to thank the other members of the CARMEL team, including Dr. Johann Borenstein, Roy Feague, Rob Giles, Kevin Mangis, Patrick Kenny, and Alex Ramos. Dr. Terry Weymouth was the faculty advisor for the team and contributed much to the effort. Support for the CARMEL team was provided by The University of Michigan College of Engineering; the American Association for Artificial Intelligence; ABB Graco Robotics, New Berlin, WI; the NTN Technical Center, Ann Arbor, MI; and

the Environmental Research Institute of Michigan (ERIM), Ann Arbor, MI. Co-authors Huber and Lee are supported by DARPA grant no. DAAE-07-92-C-R012. Purchase of CARMEL and support for research using it is provided by Department of Energy grant no. DE-FG0286NE37969, which also supported co-author Kortenkamp.

References

[1] Kenneth Basye, Thomas Dean, and Jeffrey Scott Vitter. Coping with uncertainty in map learning. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 1989.

[2] Johann Borenstein and Yoram Koren. Histogramic in-motion mapping for mobile robot obstacle avoidance. *IEEE Journal of Robotics and Automation*, 7(4), 1991.

[3] Johann Borenstein and Yoram Koren. The Vector Field Histogram for fast obstacle-avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3), 1991.

[4] Johann Borenstein and Yorem Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5), 1989.

[5] Eugene Charniak. Bayesian networks without tears. *AI Magazine*, Winter, 1991.

[6] Thomas Dean, Kenneth Basye, Robert Chekaluk, Seungseok Hyun, Moises Lejter, and Margaret Randazza. Coping with uncertainty in a control system for navigation and exploration. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1990.

[7] David Kortenkamp, L. Douglas Baker, and Terry Weymouth. Using gateways to build a route map. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1992.

[8] Benjamin J. Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, 8, 1991.

[9] Maja K. Mataric. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3), 1992.

[10] David P. Miller. A Spatial Representation System for Mobile Robots. In *Proceedings IEEE International Conference on Robotics and Automation*, St. Louis, 1985.

[11] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA, 1988.

[12] Annie Wu, Clint Bidlack, Arun Katkere, Roy Feague, and Terry Weymouth. Vision-based object pose estimation for mobile robots. In *Proceedings of the AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS '94)*, 1994.

THE GROUND VEHICLE MANAGER'S ASSOCIATE

Gary R. Edwards, Robert H. Burnard, William L. Bewley, and Bruce L. Bullock
ISX Corporation
4353 Park Terrace Drive, Westlake Village, California 91361
Phone: 818-706-2020 Email: gedwards@isx.com

Abstract

Manager's associate systems enable users to indirectly manage semi-autonomous agents to support collaborative, mixed-initiative human-computer problem solving. MAX is a software framework for building manager's associate systems. It provides an architectural model, a domain-independent software structure, tools, and reusable components for building domain-specific elements of systems used by managers to develop, execute, and analyze task plans for agents. This paper presents an overview of MAX and describes its first application: a ground vehicle manager's associate system for the management of robotic vehicles exploring a simulated planetary surface.

Introduction

The WIMP user interface (Windows - Icons - Menus - Pointing Device) was introduced by Xerox's Alto and Star and popularized by the Apple Macintosh almost 15 years ago. These products were followed by several generations of new personal computer products, each improving hardware and software functionality but not extending the standard WIMP direct manipulation interface concepts. Recently, a number of papers have appeared suggesting that a new generation of user interfaces is coming that will feature semi-autonomous agents that perform intelligent functions for the user.[1, 4, 6, 7, 9, 10, 12] Alan Kay, for example, distinguishes between manipulation interfaces and management interfaces and suggests that the interface of the future will enable users to indirectly manage agents, not directly manipulate objects.[6] Similarly, Dave Smith argues that we need delegation interfaces which support delegation of tasks to the computer, not more manipulation interfaces.[12]

The rationale for agents and agent-management interfaces is that there are many tasks for which direct manipulation interfaces do not work. Such tasks are often tedious, they may require too much of the human's time, and they usually require processes better performed by the computer. Examples include searching news and email files for information of interest to a human reader,[6] developing tactics plans for an overloaded fighter pilot,[3] and exploring a planetary surface through robotic vehicles.[5] For tasks that can and should be performed by the computer while the human is doing something the computer cannot do, agents will be needed and an interface through which the human can manage the agents will be required. We call the agent management interface the manager's associate.

The manager's associate is an extension of a concept of human-computer interaction derived from several years of research and development in a variety of domains, beginning with advanced pilot aiding in the Pilot's Associate system.[2, 3, 8] An associate system enables collaborative, mixed-initiative human-agent problem solving in application domains in which the human is unable to cope with the scale or complexity of the problem solving situation. In such domains human information processing can be overloaded by very large problem spaces, too many simultaneous activities, and too much data, but full automation is not possible because the task requires human perception, judgment, or expertise. The associate system employs models of the task and the user to provide advanced user support, including workload management, error recognition and correction, adaptive aiding, context- and user-adaptive display management, and selective task automation.[2]

This paper describes a system called MAX, a software framework for building manager's associate systems. It provides an overview of the system and then describes its application in one domain: supervisory management of robotic vehicles.

Copy right © 1993 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

The MAX Framework

Our goal in developing MAX was to leverage the lessons learned, the experience, and the architectures developed in the Pilot's Associate program to build a generic system supporting cost-effective development of manager's associate systems. This generic system must support development of applications in a range of application domains, including but not limited to avionics.

MAX is a software framework that provides an architectural model, a domain-independent software structure, tools, and reusable components for building domain-specific elements of systems used by managers to develop, execute, and analyze task plans for semi-autonomous agents. These elements can be employed to support problem assessment, focusing the user's attention on problems of interest, developing an effective problem-solving strategy, and executing the problem-solving strategy through selective task automation and human performance assistance.

Manager's associate applications are defined in MAX as a collection of Activities, which are major elements of the manager's task such as Planning, Execution, Analysis and, for control of semi-autonomous vehicles, Tele-Operation. These activities have subactivities, e.g., the Planning activity can be composed of subactivities like Assign Objectives to Resources, Plan Tasks, and Calculate Performance Measures. Each of these subactivities can in turn be composed of lower-level subactivities. For each activity, MAX provides an Activity Manager, a Data Process Manager, an Interface Manager, and a Command Monitor. The Activity Manager provides control and coordination of system resources, while the Interface Manager provides managed information to the user through a graphic user interface and provides mechanisms for the user to interact with the system. Monitors maintain and process state information of activity-specific objects and domain specific data items in a global data store. Monitors also signal alerts which may be acted upon by the associate system or the human manager, as appropriate.

A primary characteristic of manager's associate applications is that the human needs to selectively interact with large collections of data. These interactions typically include:

- Observing data, supported in MAX by modeling the classes of activities the user is involved in, locating data relevant to those activities, and presenting activity-specific abstractions of that data.

- Monitoring data, supported in MAX by data monitoring functions that apply rule-based criteria to identify the state of the data of interest to current user activities and to signal alerts based on this state.

- Selecting and applying data processing operations. For any user activity, MAX data handlers can be defined to select and apply specific operations to selected data, and can specialize the selection of operations based on the state of the data.

- Responding to alerts, supported by a mixed initiative task management scheme. For any alert condition, MAX can identify candidate responses. These responses may include:
  - Applying a data processing operation
  - Cueing a new user activity and its supporting MAX functions
  - Performing specific computations to generate additional options
  - Ignoring the alert condition

MAX decides how to handle an alert condition by applying rule-based decision models to evaluate the utility of responding to the alert, the legality of each known option, the utility of each legal option, and the utility of automating the function for the user. This process allows MAX to consider a number of factors in deciding what options to present to the user, whether any should be suggested as the "best" action for the user, or whether any options should be automatically pursued without user intervention. The factors that drive this mixed-initiative reasoning include the priority of the problems the user is facing, the user's workload, the user's preferences for selecting specific options, the user's preferences for various levels of automation, and the utility of applying specific options to the problem at hand.

The MAX framework provides a set of supporting software to enable and simplify

the construction of domain-specific applications that exhibit these capabilities. In its current form, the MAX framework provides this support by defining three categories of application developer tools:

- A set of "standard" components (and tools to define them) that can be used to construct MAX application components and specify their run-time interactions in a high level representation. These include "hooks" and tools for specifying domain-specific interface components, data definitions, and processing procedures, and "hooks" to specify user tailoring of the application's behavior.
- A collection of procedural attachments to those components that implement high level specifications of control interactions as low-level run time control decisions.
- A collection of "default" interface components.

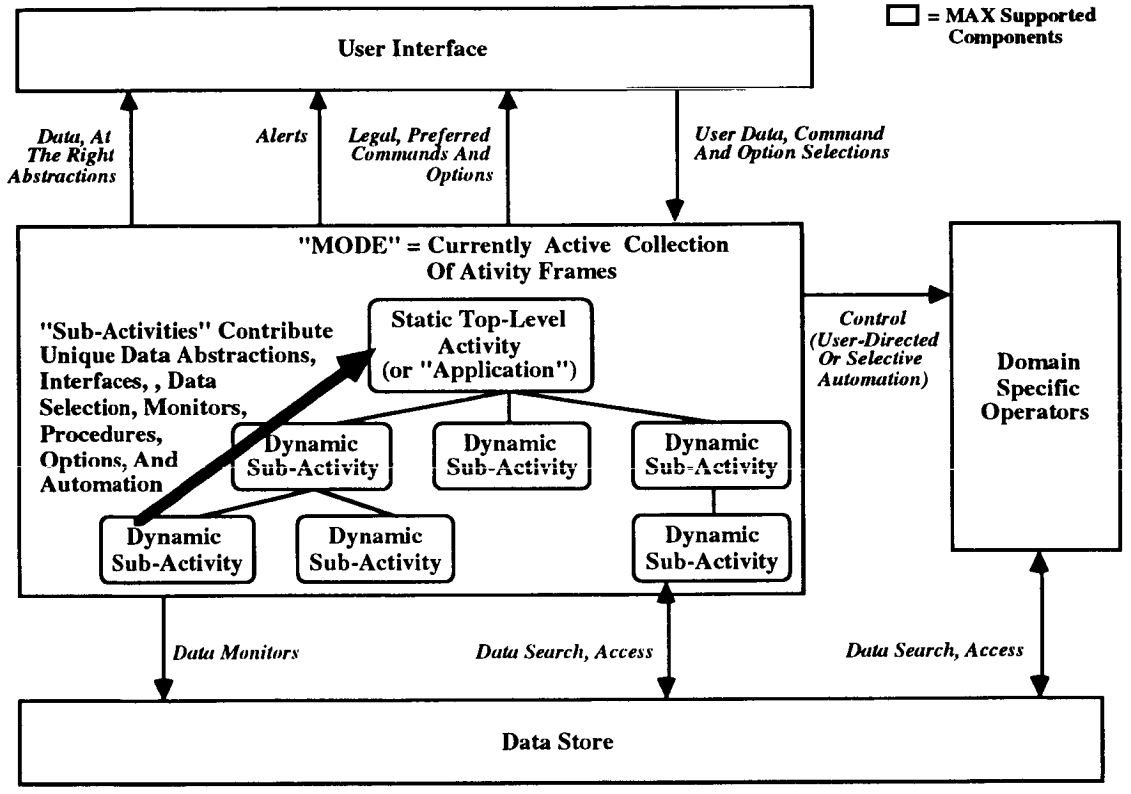


Figure 1. Run-Time Anatomy of a MAX Application

System Architecture

MAX is centered on the concept of Activities. The application user, at any point in time, is operating in a mode comprised of a currently active collection of activities. Each activity contributes interfaces, specialized tools, data of interest, and data monitors to the user's environment while operating in that mode. Figure 1 illustrates the interaction among activities, composing and maintaining a unique combination of sub-activities at run time to respond to both the dynamically changing set of problems confronting the user and the user's own direction for reacting to those conditions.

Each activity in MAX is composed of a few basic components, repeated as needed to define a complete activity. These components, along with their primary interactions, are shown in Figure 2. The top level MAX activity, upon activation, creates its interface manager and data manager. The interface manager in turn creates appropriate

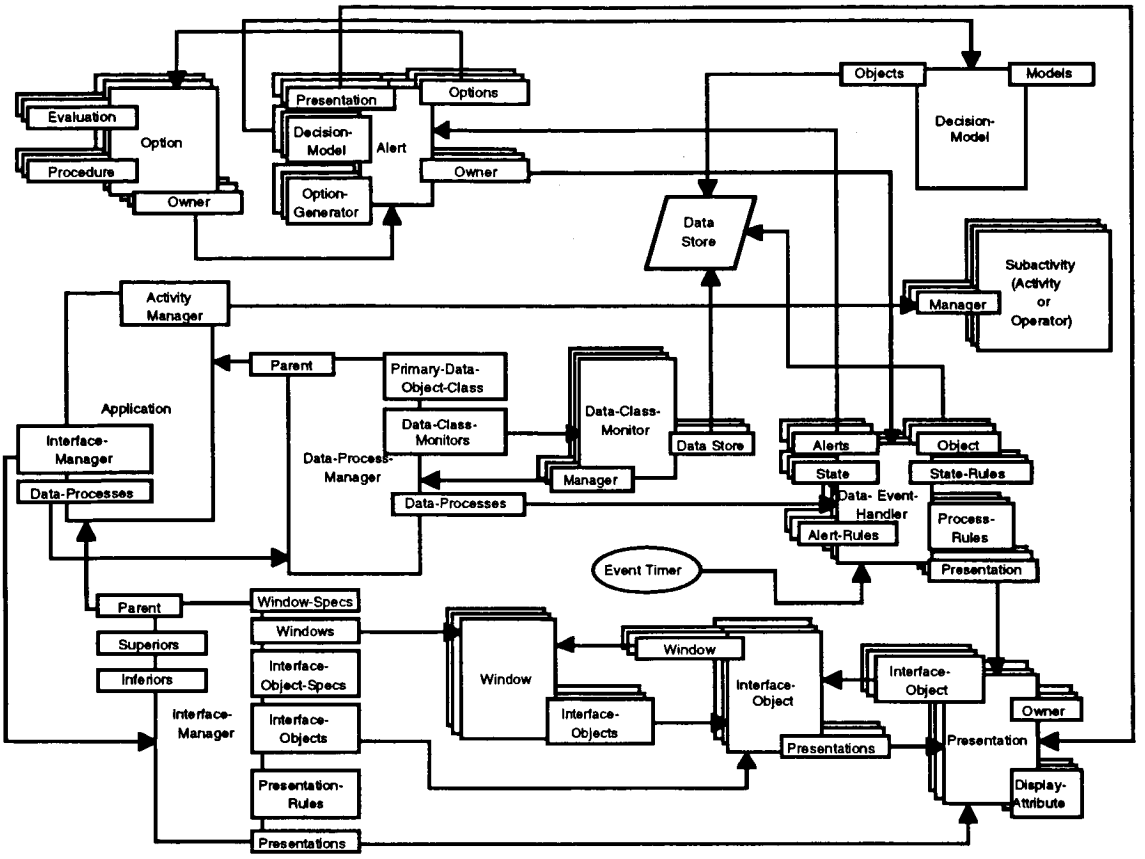


Figure 2. Anatomy of a MAX Activity Frame

instances of MAX's built-in interface objects (windows, menus, buttons, presentations, etc.) along with those domain-specific interface components specified in the activity definition.

The activity also creates a data manager, which establishes data class monitors to detect objects of interest in the data store. Each object of interest found by these monitors (either immediately or at any time in the life of this activity) causes a data-event-handler to be created, and a data-instance-monitor to be established. This instance monitor detects any changes in the object of interest, and triggers data change events, These in turn cause the data-event-handler to reevaluate the object of interest, and may cause it to either trigger associated data processing operators, or may trigger an alert condition. For data whose impact on the problem may change over time rather than over changes in value, a clock-driven event timer is provided to re-trigger data evaluation.

In an alert condition, an alert is asserted. The alert in turn generates a set of possible options to deal with that alert. Options consist of three classes of operations: data processing procedures; procedures that generate new options; and invocation of sub-activities to help the user solve the problem. Next, the alert determines which options are legal in the current context, and then evaluates the utility of applying each legal option. For the best options, the alert also evaluates the utility of automating the option's execution, as opposed to presenting it for the user's consideration. In each of these steps, "evaluating utility" employs rule-based decision models that consider problem characteristics, the user's current problem solving workload, the user's personal preferences among options and for his desired level of automation support, and models of the capabilities and current state of the MAX application.

Implementation

MAX was developed in Macintosh™ Common Lisp, version 2.01p3 and Expertelligence's Action!™, version 3.0. It runs on a Macintosh IICI with 20MB of RAM, an 80MB hard drive, and a 13 " or larger color or gray scale monitor. The system can run in other Macintosh environments, including a Powerbook™, with modification of the map displays.

The GVMA

MAX is designed to support a wide range of manager's associate applications. The choice of the Ground Vehicle Manager's Associate (GVMA) was made in collaboration with NASA Ames to support a successful demonstration of simulated planetary exploration using IS Robotics, Inc. robotic vehicles.*

The GVMA has three main activities, the mission planning activity, the mission management activity, and the vehicle tele-operation and video survey activity. A robot simulation activity was also created to provide an internal vehicle simulation that can be executed from within the GVMA.

Monitoring the activities of all vehicles, the system alerts the manager to events requiring human attention, provides options for human action, and delivers vehicle commands that implement options invoked by the manager. The manager monitors the sensors and can directly control each vehicle through the Tele-Operate activity window, which includes a live video feed from the vehicle. The overall activity of the vehicles is displayed in the Mission Management window. The GVMA observes the performance of each activity against a mission plan. When a situation triggers a monitor, the GVMA displays an alert and suggests actions to the manager through an Alert window. If the manager invokes one of the suggested options, the GVMA executes the selected action. Figure 3 shows an example in which an alert has signaled readiness to perform a search activity at a location on the planetary surface. At this point, the manager can tell the system to invoke the search immediately, wait for the start time specified in the mission plan, or wait for further direction. If the activity is invoked, the GVMA will cause the vehicle to proceed to the search area via the specified waypoints.

Assessment

The GVMA demonstration showed that MAX provides a framework for building an associate system for managers of multiple vehicle missions. We plan to apply MAX to the development of manager's associate systems in different domains in the near future.

Using MAX, the GVMA was developed in two person-months, including interfacing to the robot vehicles. We did not attempt to build a non-MAX GVMA in order to compare development time and difficulty, but based on our experience we believe that an application of the complexity of the GVMA would have required at least six person-months if developed from scratch.

Although this suggests that MAX has value in developing manager's associate applications, it is not a finished product. We think that it was helpful in developing the GVMA, but enhancements are required. Facilities for configuring communications interfaces and for developing task, environment, and user models work, but they are difficult to use. In addition, the software is not as robust as it should be. MAX supported the development of the GVMA, but it has not been tested in a variety of applications, and such testing will almost certainly reveal undetected bugs. Finally, more effort needs to be invested in developing planning and decision analysis functionality. The decision analysis module works, but it is fairly primitive and decision models are difficult to build and revise. The current planning module requires the user to develop plans using a very simple editor, and there is no replanning capability. Determining the effectiveness of the GVMA and other manager's associates is problematic. As noted by Sheridan,[11] the objective function of supervisory control is not fixed and cost and complexity make it extremely difficult to conduct controlled experiments in this domain. It is certainly possible to demonstrate the superiority of the GVMA over manual control, but

*IS Robotics, Inc., Twin City Office Center, Suite 6, 22 McGrath Highway, Somerville, MA 02143.

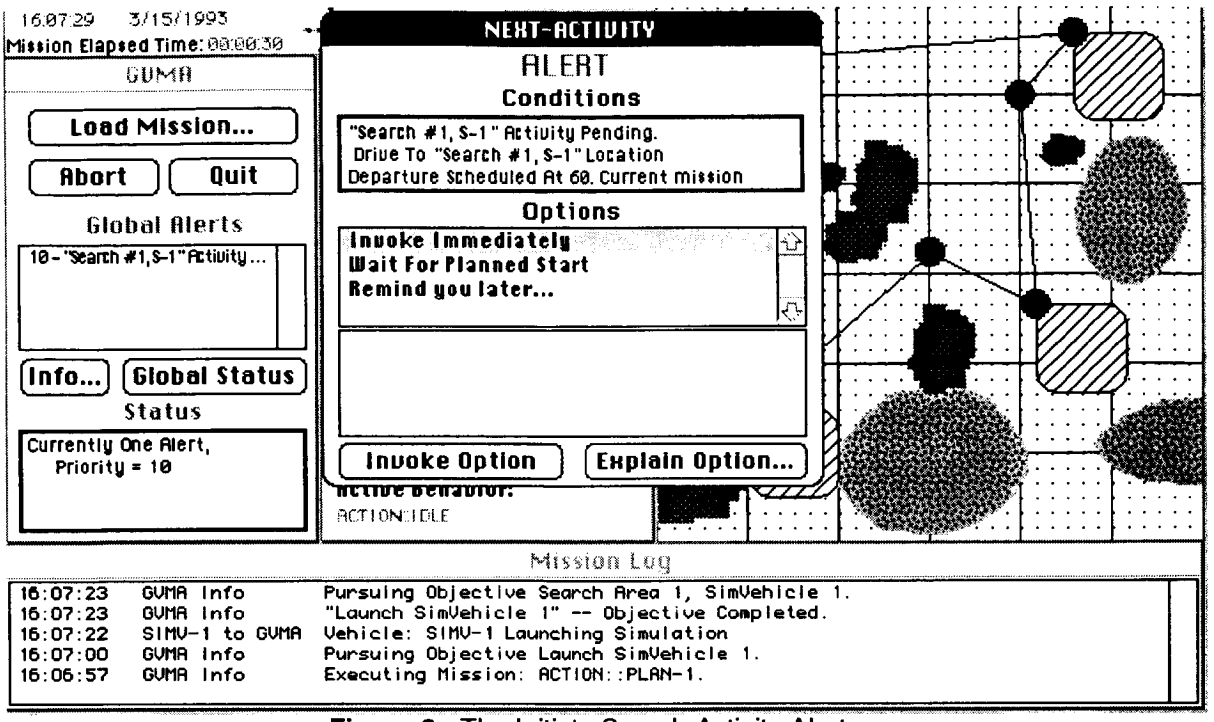


Figure 3. The Initiate Search Activity Alert

understanding the relative effectiveness of the many forms and combinations of associate features and behaviors is another matter. We are investigating ethnographic and usability inspection methodologies as adjuncts to the methods of experimental psychology.

Summary and Future Work

Agents and manager's associates are required by many important new applications, and we will be seeing commercial implementations of such interfaces in the near future. The first products will probably support information access applications that employ agents in storing, retrieving, manipulating, and understanding massive amounts of information. The management of intelligent devices such as remote vehicles, manipulators, and instrumentation will be another important application. These applications will require a manager's associate, and we believe that cost-effective development will require a framework like MAX.

As noted in the discussion of effectiveness, MAX is not a finished product. We plan to continue work on the system, improving it as we use it build new applications. First steps are enhancement of the planning and decision analysis functionality. We are also working with IS Robotics, Inc. to extend the GVMA and interface it to new micro-robot platforms.

Acknowledgements

MAX and the GVMA were developed with support from NASA Ames Research Center Small Business Innovative Research contracts. IS Robotics, Inc. supplied the robotic vehicles, and Dr. Carl Friedlander was responsible for developing interfaces and on-board behavior code. Dr. David Korsmeyer was responsible for management of the GVMA demonstration at NASA Ames Research Center.

References

1. Card, S.K., Robertson, G., and Mackinlay, J.D. The Information Visualizer: An information workspace. In *Proceedings of SIGCHI '91*, 1991, 181-188.

2. Edwards, G.R. and Geddes, N.D. Deriving a domain-independent

architecture for associate systems from essential elements of associate behavior. In *Proceedings of the First DARPA Workshop on Associate Systems Technology*, 1991.

3. Fields, C.I.. & Kushner, B.G. The DARPA Strategic Computing Initiative. In *IEEE Proceedings of COMPCON*, 1986.

4. Fischer, G., Grudin, J., Lemke, A. C., McCall, R., Ostwald, J., Reeves, B. N., and Shipman, F. Supporting indirect, collaborative design with integrated knowledge-based design environments. *Human Computer Interaction*, 7, 3, 1992, 281-314.

5. Geddes, N. and Hoffman, M. Supervising unmanned roving vehicles through an intelligent interface. In *Proceedings of SOAR '91*, 1991.

6. Kay, Alan. User interface: A personal view. In B. Laurel (Ed.), *The Art of Human-Computer Interface Design.* Reading, MA: Addison-Wesley Publishing Company, Inc., 1990, pp. 191-207.

7. Laurel, Brenda. Interface agents: Metaphors with character. In B. Laurel (Ed.), *The Art of Human-Computer Interface Design.* Reading, MA: Addison-Wesley Publishing Company, Inc., 1990, pp. 355-365.

8. Lizza, C. and Friedlander, C. The Pilot's Associate: A forum for the integration of knowledge based systems and avionics. In *Proceedings of NAECON*, 1988.

9. Nielsen, Jakob. Noncommand user interfaces. *Communications of the ACM*, 36, 4, (April 1993), 83-99.

10. Robertson, G., Card, S.K., and Mackinlay, J.D. Information visualization using 3D interactive animation. *Communications of the ACM*, 36, 4, (April 1993), 57-71.

11. Sheridan, T.B. Supervisory control of remote manipulators, vehicles and dynamic processes: Experiments in command and display aiding. *Advances in Man-Machine System Research*, 1, 1984, JAI Press, 49-137.

12. Smith, David C. Common elements in today's graphical user interfaces: The good, the bad, and the ugly. In *InterCHI '93 Conference Proceedings*, Amsterdam, April 1993, 470-473.

AIAA-94-1250-CP

Deictic Primitives for General Purpose Navigation*

Jill D. Crisman, Ph.D.
Robotic and Vision Systems Laboratory
Departent of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115

Abstract

We are investigating visually-based deictic primitives to be used as an elementary command set for general purpose navigation. Each deictic primitive specifies how the robot should move relative to a visually distinctive target. The system uses no prior information about target objects (e.g. shape and color), thereby insuring general navigational capabilities which are achieved by sequentially issuing these deictic primitives to a robot system.

Our architecture consists of five control loops, each independently controlling one of the five rotary joints of our robot. We show that these control loops can be merged into a stable navigational system if they have the proper delays. We have also developed a simulation which we are using to define a set of deictic primitives which can be used to achieve general purpose navigation. Encoded in the simulated environment are positions of visually distinctive objects which we believe will make good visual targets. We discuss the current results of our simulation.

Our deictic primitives offer an ideal solution for many types of partially supervised robotic applications. Scientists could remotely command a planetary rover to go to a particular rock formation that may be interesting. Similarly an expert at plant maintenance could obtain diagnostic information remotely by using deictic primitives on a mobile platform. Moreover, since no object models are used in the deictic primitives, we could imagine that the exact same control software could be used for all of these applications.

1. Introduction

We are developing a robot architecture which uses a natural deictic interface that allows the user to point out targets to the system. To operate a deictic mobile robot, the user would select a target in a video image and then issue a command such as "approach that" or "pass to the right of that" where 'that' is the target selected in the video image. In this paper, we describe the robot architecture that we are using for this deictic system. We also describe our simulation environment that we are developing to explore the definition of a set of deictic primitives to be used for general purpose navigation.

This work is important since the elementary deictic primitives give researchers a novel way to think about programming robot systems. Most robots are controlled by specifying a target in geometric terms, for example as a Cartesian position and orientation (e.g. 'go to 20m, 12m, and face 10 degrees') or as a location on a map. On the other hand, deictic primitives would involve a user pointing out a sequence of

* This work is supported by the National Science Foundation under grant number IRI-921056. This work is also aided by the donation of a Cognex 4400 Machine Vision System by Cognex Corp., Needham, MA.

Copyright c 1993 by the American Institute of Asronautics and Astronautics, Inc. with permission

visual targets and the robot moving relative to those targets. We believe that this type of programming interface is more natural for humans since people tend to move relative to what they perceive. For example, we would 'walk to the doorway' rather than 'walk forward 10 feet'. As our work progresses in the future, we will add object models so that our system would be able to 'approach the doorway'. Therefore, we believe that deictic commands would be a more natural method for people to interact with a mobile robot system.

This deictic interface is very different than interfaces to traditional mobile robots. Many robots are controlled by specifying a target location in geometric Cartesian coordinate with respect to an initial robot location. In this case, the robot must keep track of its location in order to know if it has reached the goal location. Other mobile robots navigate with respect to a map of the environment where goal locations are specified by a geometric coordinate on the map. The robot must continually track its position with respect to the map to determined if it has obtained its goal. Still other robots navigate to target objects which have pre-stored models so that the robot can identify landmarks. In all of these traditional approaches to interfacing with the robot, environmental knowledge must be encoded geometrically for the system to operate.

Our deictic system is very different in that the robot only needs to keep track of the destination object in it video field. Since target tracking is more robust than object identification, the processing time of our system is decreased. The robot does not need to keep track of its location with respect to a global map, therefore our system is not susceptable to position tracking errors. We take advantage of movable camera systems to simplify our robot control architecture.

This deictic interface for semiautonomous robots has many applications, especially in exploratory robots. Scientists can control a planetary rover by selecting a location of interest in the video screen and commanding the robot to go to that area. Underwater robots can be controlled with lower bandwidth communications than is typically necessary for remotely operated vehicles. Moreover, semi-autonomous robots have applications in aids for the handicapped.

In this paper, we overview the robot architecture which uses five feedback control loops to control the motion of the robot. We show that with the time constants on the feedback loops that this system can provide smooth and stable motion of all joints of the robot. We also present our initial work on a simulator for exploring the definition of a set of deictic primitive commands. We show the results of this simulation for a series of approach commands.

2. Related Work

Developing mobile robot systems based on traditional computer vision and robotics paradigms requires the use of an *a priori* object model for the goal and a reference coordinate frame [16] [20]. The vision system identifies the goal in the scene by using the *a priori* object model provided. The object positions and orientations are perceived in the camera coordinate frame and must be transformed into the reference coordinate frame and added to the world model. Other sensor modules add information to the world model. Motion decisions for the robot system are made by a path planning module using the most recent information from the sensors which has been integrated into the world model. As the robot moves, the system must record and update the robot's position within the world model. This system has been used in many robotic systems including [21] [11]. This traditional solution is somewhat limited since it assumes that prior object models are available, which is often not the case in applications such as planetary exploration and household robotics.

Similar systems, for example [13], construct a world model without having the *a priori* object models. However, the world model construction process is computationally very

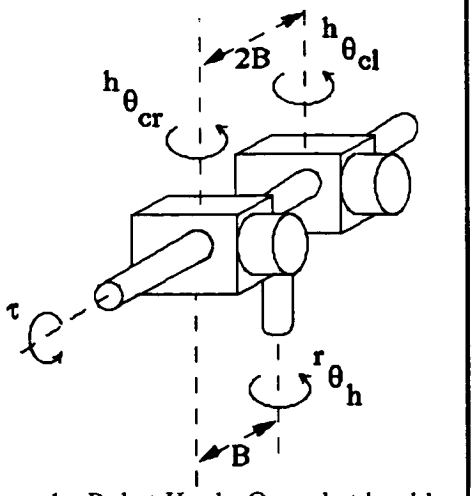


Figure 1: Robot Head. Our robot head has four joints. The first joint controls ${}^{r}\theta_{h}$, the pan of the head with respect to the robot base. The second joint controls the tilt, τ, of the cameras. The third and fourth joints control the pan of the cameras

expensive. These systems require calibration between the camera system and the robot, a localization routine so that the robot can identify its location with respect to the local map (so that the world model can be integrated over time), and a good kinematic and dynamic model of the robot system. The calibration, kinematic, and dynamic models always have associated with them some approximation errors. Motion planning, which is done on the world model, can become difficult as the robot modeling errors accumulate.

Visual servoing techniques have been proposed to eliminate the geometric dependence of the motion commands. Rather than directing the robot to a destination location, the robot is instructed to maintain its visually apparent position with respect to an object using dynamic visual feedback. Robot manipulators with a camera mounted on the arm can now track specific objects in 3-D space [22] [10] and navigation systems can track pathways [6] [9]. These systems work in real-time by tracking a specific visual feature rather than reconstructing a complete 3D description of the world.

Other researchers have abandoned traditional methods and instead have promoted behavior-based robotic architectures and local path planning algorithms [1] [3] [4] [12] [19]. These systems tend to use a distributed computer system to acheive tightly coupled control loops between the sensing and actuation. Therefore these systems have better reaction times in the presence of moving objects. Ultrasonic sensors are a common choice to provide fast obstacle detection [2] [14].

Our system currently uses a simple and fast method for determining the motion of the robot and most closely resembles these behavior based systems. Therefore our system is able to react quickly to a moving or newly detected obstacle. We use a visual servoing technique to position the gaze of each camera directly at the target. The mobile robot then moves in the gaze direction of the cameras if the pathway is clear of obstacles. Otherwise it moves around the obstacle and continues seeking the target.

3. Mobile Robot Hardware

Our experimental equipment consists of a mobile robot base with a ring of ultrasonic sensors, an active robot head, and a high speed video processor. The active robot head has four controllable motions. The robot head carries two cameras and controls the pan of each camera individually and it controls the tilt and pan of the pair of cameras, as shown in Figure 1. This platform is similar to those described in [5], [15], and [17]. The platform was constructed such that the pan and tilt of the cameras occur approximately about the focal point of the cameras. A Cognex 4400 Machine Vision system is currently handling the real-time video processing of the cameras. The active camera head is mounted on a mobile robot platform with a ring of 24 ultrasonic sensors. Each ultrasonic sensor can determine the distance to the closest object in a 30^{o} field of view.

4. System Architecture

Our goal is to achieve fast, reliable pursuit of a target while avoiding obstacles in the path. Our system includes three components: a target tracker, obstacle detector, and mediator as shown in Figure 2. The target tracker follows the target location selected by the user and reports the angle and distance of the target to the mediator. The active robot head is used to simplify the target tracking task. The obstacle detector reports the measurements from the ultrasonic sensor ring. These measurements are the distance to the closest object within the field-of-view of each sensor as a function of angle from the robot. The mediator then determines the speed and steering angle of the robot. In the following subsections, we describe in more detail the three components of this system.

4.1. Tracker

The tracker is responsible for reporting the angle and distance to the target. Since we are focusing on a video interface, we will be using targets from video images from the stereo cameras. We are using stereo cameras to determine the distance to the target. While determining the distance to a stationary target is possible from a moving platform with a known motion, we do not assume that the target is stationary nor that the motion of the target is known. As the robot and target are moving, the tracker must determine the location of the target in the image. Since the target can easily move outside of the field of view of the cameras, we use an active robot head to keep the target in sight and thus to simplify the tracker.

The tracker operates as four independent controllers, one for each motion of the camera head: right camera pan, left camera pan, head pan and tilt (see Figure 1). The target is first located independently in each stereo image. The camera pans, θ_{cl} and θ_{cr}, and the head tilt τ are used to move the cameras such that the position of the target appears in the center of the stereo images. The head pan is independently controlled to try to face the cameras directly at the target. The angle to the target can then be directly measured from the pan of the robot head. The angles of the stereo cameras with respect to the robot head can be used to compute the distance to the target. For more details of this controller see [7] and on video tracking [8].

4.2. Obstacle Detection

The sonar system is responsible for reporting the locations of obstacles surrounding the vehicle. In a typical ultrasonic system, each

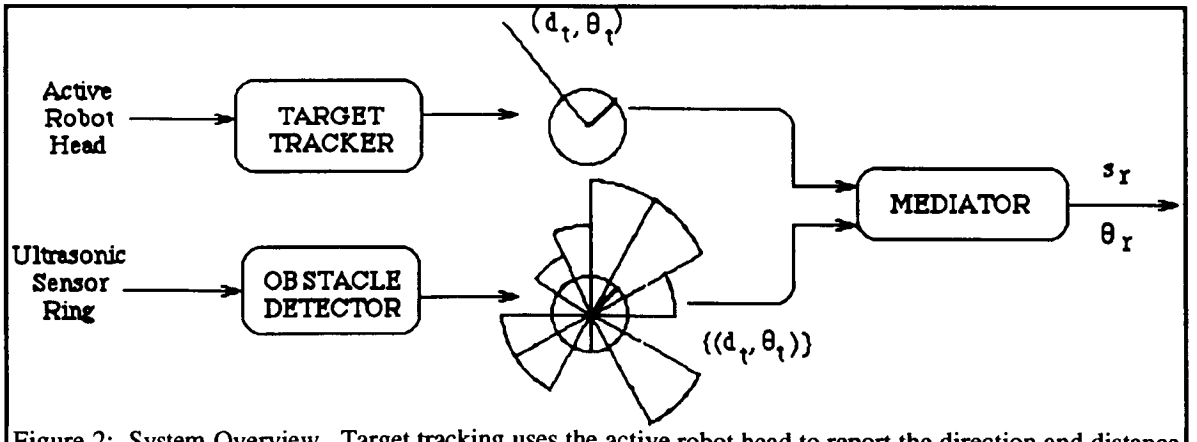


Figure 2: System Overview. Target tracking uses the active robot head to report the direction and distance of the target relative to the mobile robot base. Obstacle detection reports the distance to the closest object within the field-of-view of each sonar sensor. The mediator picks the best speed and steering angle commands for the mobile robot base.

sonar covers a 30° field-of-view. The object which is closest within this field is detected by the sonar. The sonars are spaced in a ring around our platform. The mediator receives the result of each sonar individually. These readings can be thought of as the cost of the robot traversing in that direction.

4.3. Mediator

The mediator decides the steering and speed commands that will be sent to the mobile robot. The tracker reports to the mediator the current direction and distance to the target. The obstacle detector determines a radial map of distances to obstacles surrounding the vehicle (see Figure 2). Interestingly, we found that the mediator need not be complex to steer the robot successfully.

Consider that the robot can only steer within the resolution that it can sense. Therefore, to track the target in an image, the robot can steer according to the resolution of the pixels in the image. However, if obstacles are detected, the robot only knows that an obstacle appears within a 30° field-of-view. Therefore, the robot can only steer in 30° increments. Each ultrasonic reading corresponds to a steering direction. If an ultrasonic sensor detects an obstacle, then the robot should not steer into the 30° field-of-view of the detecting sensor.

If there are no obstructions in the direction of the target, then the robot pursues the target direction. If there is currently an obstruction in the direction of the target, the mediator will select the closest open steering angle to the target.

The mediator also considers the closest obstacle and the distance to the target when selecting the vehicle speed. The speed is inversely proportional to the distance to the closest object. We pursue the target to within a fixed distance. For safety reasons, the robot's speed is also clipped to a maximum value.

4.4. Simulation

To show the competence and stability of the system we have simulated a robot motion model to test our navigation algorithms. To ensure a realistic simulation, we have modeled each motion of the robot as a second-order system. The motion of the robot joints is modeled as a damped response to the desired motion commands issued by the mediator.

At each step in our simulation, two camera images and 24 ultrasonic measurements are taken of the environment. We assume that these measurements are relatively accurate. We completely model the limited field of view of the cameras and the quantization of the camera measurements. We also add random noise to these measurements. The ultrasonic measurements also have noise added and we model a 30° field-of-view of the ultrasonic sensors.

The simulation keeps track of the motion of the target and the motion and orientation of the robot with respect to a world coordinate frame. Notice that in our architecture, the robot does not know about a world coordinate frame since it has no world model. The robot only concentrates on pursuing the target location and it considers its location in the world irrelevant. For the purpose of display and sensor input computations, we represent locations of objects, targets, and the robot with respect to a world coordinate frame. Our simulation is two-dimensional, ignoring the z axis. Therefore, the tilt of the camera head is not simulated.

In the following subsections, we describe the simulation of the camera input, the sonar readings, and the motion model of the robot.

4.4.1. Camera Pan and Tilt Simulation

For our simulation, we currently do not model projection, back projection, and camera measurements. Instead, we compute

the desired angle for the camera pans by transforming the position of the target to the camera frame. The transformation between the camera frame and the world coordinate frame is updated as the robot moves.

4.4.2. Ultrasonic Measurement Simulation

The obstacles in our simulation are represented by their corner locations. For each corner of an object, the position of each corner is transformed to the coordinate frame of the robot. We then compute the angle to this location to determine in which of the ultrasonic measurements this corner will appear. If the new distance, with additive noise, is less than the current minimum distance know by that sensor, then the sensor measurement is updated Given the range ultrasonic sensor in the ring effected by each object allows us to compute the intermediate sonar values.

4.4.3. Motion Control

We model each joint motion as a second-order system. We assume that the joint controller is critically damped and that the discrete inputs from the computer controller are modelled by step input functions. This type of motion is achieved by using a proportional-derivative (PD) controller. These PD controllers have been successful in controlling the vergence of stereo cameras on a robot platform [18]. The motion response to the desired input is shown in Figure 3. The equations of the response function is:

$$\theta(t) = \theta^d (1 - \exp(t/\tau))$$

where t is reset to zero when θ^d changes. θ^d is the desired angle of the joint that is computed by our joint motion algorithms described previously. θ^d is a piecewise step function since it is being computed by a discrete controller. τ is the time constant of the system which controls how fast the joint can track the desired input. We also limit the velocity of each joint and we insure that the motion of each joint stays within its range.

Our current parameter values for the time constant and maximum velocity for each joint is summarized below:

$\tau_{cr} = 50$	$\lvert\omega_{cr}\rvert_{max} = 90$ deg/sec
$\tau_{cl} = 50$	$\lvert\omega_{cl}\rvert_{max} = 90$ deg/sec
$\tau_h = 10$	$\lvert\omega_h\rvert_{max} = 60$ deg/sec
$\tau_r = 5$	$\lvert\omega_r\rvert_{max} = 30$ deg/sec

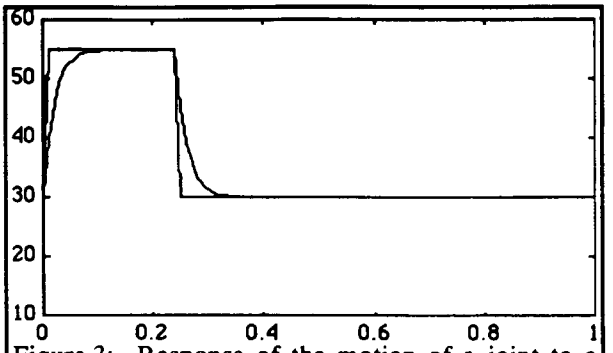


Figure 3: Response of the motion of a joint to a specified input.

4.5. Results

We have run the simulator on numerous examples and we show a couple of results here. In all attempted scenarios, we have successfully arrived at the target location without colliding with obstacles. In the first example, we assumed a stationary target at location (10,7) with respect to the initial robot frame (see Figure 6.) Recall that the x coordinate of the robot frame specifies its direction of motion. Since our slowest time for processing a single frame was 100 milliseconds, we used this time as the sampling period of the system. We assumed that the vehicle could travel a maximum of 3 meters/second.

We present a test sequence where the target is at the limit of the cameras' field-of-view. Therefore, the desired pan of the cameras will be at its largest possible value. We demonstrate to show that the system is stable and controls the head and robot motions smoothly even given the largest step input to the system.

Figure 4 show the motion of the left and right cameras with respect to time. As the robot begins it journey, the cameras first notice that the target is about 40° to the left of the robot. The cameras begin to pan to the target and the head begins to pan to face the cameras toward the target. The system normalizes when the angle of the head and the cameras is small. In this case, the angles between the left and right cameras will become equal in magnitude and opposite in sign. This occurs at about 1 second. This angle magnitude remains close to zero while the target is far away, but as the robot approaches the target the cameras begin to verge. The magnitudes of the two camera angles are still about equal which indicates that the pan of the head is still correctly facing the target. When the mobile robot arrives at the target location at about 4 1/2 seconds the left and right camera angles are verged at -60° and 60° respectively. This angle can be used to compute the distance to the target. When the simulation was allowed to run to acquire the target, the camera angles became -90° and 90° respectively.

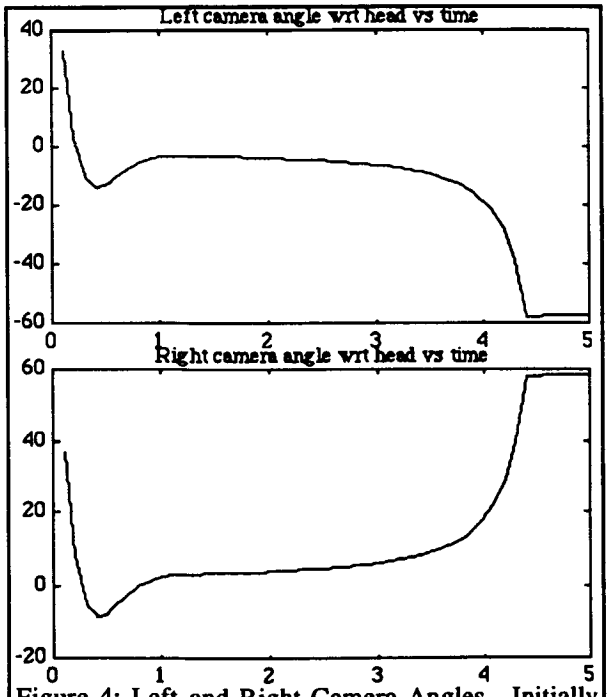


Figure 4: Left and Right Camera Angles. Initially the robot and the camera head are facing away from the target at about an angle of -40°. The cameras and pan stabilize on the stationary target location at about 1 second. From then on the magnitudes of the camera angles are approximately equal. The robot arrives close to the target at approximately 4.5 seconds.

Figure 5 shows the angle of the camera head over time. Confirming what we noticed in the camera angles, the pan motion becomes zero as the cameras are stabilized on the target location at about 1 second. Notice that when the cameras first observe that the target is at 40° the robot head begins to pan to face the cameras toward the target. The pan of the head never gets all the way to 40° since the robot itself also turns in the direction of the pan. As the system stabilizes, the pan of the head is zero since the robot is facing the target.

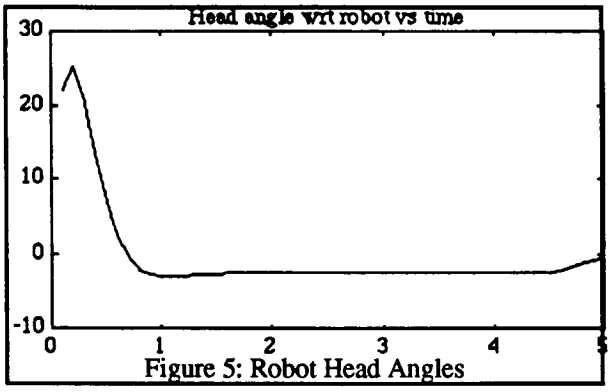


Figure 5: Robot Head Angles

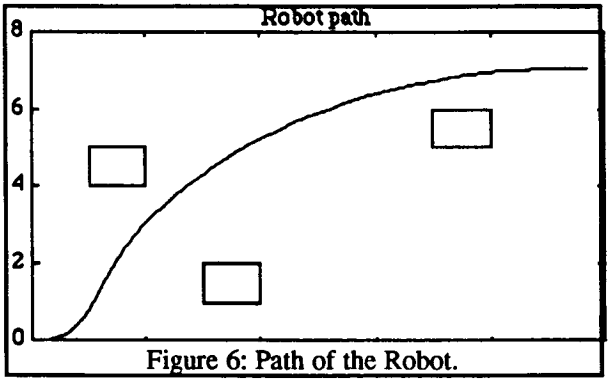


Figure 6: Path of the Robot.

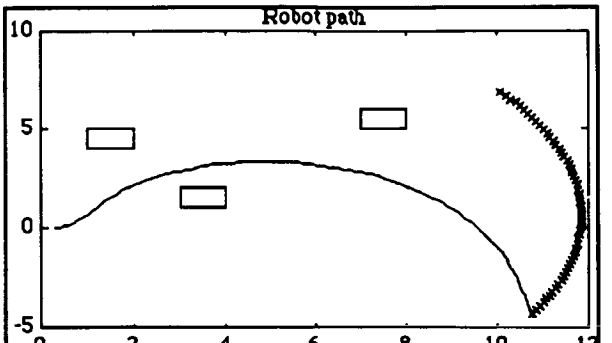


Figure 7: Robot path in Pursuit of a Moving Target

Figure 6 shows the path of the robot to the stationary target at (10,7). The robot avoids a couple of obstacles that were placed close to the straight line path to the goal. Notice that the motion of the robot corresponds to smooth forward trajectories that would be possible with a nonholonomic robot that would be steered similarly to an automobile.

Finally, Figure 7 show the path of the robot tracking a moving target. The target is following a circular path with a changing radius. The target locations, denoted by an 'x', begin at position (10,7) and end at position (10.4, -4.75). The interesting thing is that even though the robot is not estimating the motion of the target, the path developed by the visual pursuit algorithm seems to anticipate the new location of the target and correctly intercepts it.

4.6. Discussion

In our system, the motion of the camera head, panning the two cameras toward the target, is a redundant motion with the steering of the robot. This motion is necessary to allow the robot to freely manuever around obstacles without allowing the target to move outside the field-of-view of the cameras at the maximum camera angles. This gives the robot the freedom to track a target that may even move behind the robot.

The architecture is very simple and provides for much of the navigational and path planning abilities necessary in the system. Unlike other path planning research, we are not focusing on singular conditions in the path planning (e.g. trapping in 'U' shaped obstacle on path to the goal.) This is because our system inherently has a human in the loop, who can select a new intermediate target to move the robot away for the trap.

We discovered that the all the joint motions will oscillate if the response times of the camera pans, head pan, and robot turning are the same. Smooth paths were generated and smooth positioning of the cameras were obtained only if the response of the camera pans are faster than the response of the head pan which in turn is faster than the response of the robot.

5. Deictic Command Simulation

We have also extended our previously described simulation to explore the deictic primitives that are necessary to perform a general purpose navigation. Our goal is is to catalog a large number of environments and the visually interesting or trackable features of the environment. Each environment also has a set of possible goal locations. Using this simulator, we test if the robot can traverse from all starting locations to all possible goals using deictic commands in reference to the visually distintive to the targets.

We read polygonal environment descriptions from an input file. We also mark on these files, objects in the environment which we feel are easily trackable by our video system. We currently have descriptions of a standard living room and the third floor corridors of one of the buildings at Northeastern University.

Currently, we have implemented an approach command where the robot directly approaches the target location. We show examples of paths taken by our robot when commanded to approach a sequence of

targets. The data depicts the corridors of the Northeastern University engineering building and we navigate to targets which we feel are trackable by video systems in the corridors. In Figure 8, we show the robot navigating in the corridors from just outside the elevators on the third floor of our Snell building to the doorway between Snell and Dana. The robot is issued three approach commands: The first target is the sign on a vending machine near the end of the first corridor. The second commands approaches a doorknob on the door at the start of the second corridor. The final command approaches the sign on the door at the end of the corridor.

In Figure 9, the robot goes to an office in Snell, again from outside the elevators. The robot first approaches the fire alarms mounted on the wall to the left near the end of the first corridor. Then it approaches a sign on a door office to round the corner. A second alarm becomes the next target, and finally, the poster in the office is used to navigate the robot into the office.

6. Conclusions and Future Work

Our initial work on integrating an active robot head into a navigation scenario has been extremely promising. We have shown that a simple, 'follow your eyes' scenario is sufficient for tracking a moving target. In our situation, we do not plan extensive paths through the field of obstacles but we rely on a low resolution sonar sensor to detect obstacle locations. The motion of the joints on the robot head is smooth and can react to step changes in the target location. We enforce in our simulation a reasonable model of the response of the mechanical systems and the limitations of velocity and acceleration. Because of this modeling of the robot motion latency, the simulation produces realistic paths of the robot.

We are implementing our algorithms on our hardware platform and intend to develop algorithms for obstacle detection using the active robot head. We will test this algorithm extensively to determine what steps we will need to improve the algorithm to acheive better performance in many environments. We will also begin working on vision algorithms that can robustly track many targets. We want to develop a number of visually directed commands useful for general navigation. Later, we will extend this work to include targets and orientation constraints. We hope to eventually develop a set of visual commands for manipulation as well.

Not only does this system provide solutions in current semi-autonomous applications, it is also an alternative philosophy for developing fully-autonomous, general-purpose mobile robot systems. Many researchers are developing autonomous mobile robots which can navigate in limited situations, for example road-following or corridor tracking. Their philosophy is to merge autonomous systems performing specific tasks and to derive a general purpose autonomous system. We, on the other hand, are developing a robust mobile robot which can navigate in general situations. To make general mobility possible, our system will rely on more human interaction than typical mobile robot systems. Over time we will decrease the amount of user interaction by adding general environmental knowledge to the system thereby increasing the autonomy of the system. This will result in systems that are easily configured to a number of applications including underwater and space exploration, flexible manufacturing, and robotic wheelchairs.

References

[1] R.C. Arkin, "Motor Schema-Based Mobile Robot Navigation," *Inter. Journal of Robotics Research*, vol. 8, no. 4, 99-112, Aug 1989.

[2] R.C. Arkin, "Navigational Path Planning for a Vision-Based Mobile Robot," *Robotica*, vol. 7, 49-63, 1989.

[3] J. Borenstein and Y. Koren, "Teleautonomous Guidance for Mobile Robots," *SMC*, vol. 20, no. 6, 1437-1442, Nov/Dec 1990.

[4] R. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Trans. Robotics and Automation*, vol. RA-2, no. 1, 14-23, 1986.

[5] C.M. Brown, "The Rochester Robot," Tech. Rep., Computer Science, 257, Sept 1988.

[6] J.D. Crisman, "Color Vision for the Detection of Unstructured Roads and Intersections," Ph.D. thesis, Carnegie Mellon University, Dept. of Electrical and Computer Engineering, May 1990.

[7] J.D. Crisman, Y. Du, and M. Cleary, "Adaptive Control of Camera Position for Stereo Vision," in *Optics, Illumination, and Image Sensing for Machine Vision VIII*, SPIE, Boston, MA, September 1993, invited paper.

[8] J.D. Crisman and Y. Du, "Generic Target Tracking Using Color," in *Intelligent Robotics and Computer Vision XII: Active Vision and 3D Methods*, SPIE, Boston, MA, September 1993, invited paper.

[9] E.D. Dickmanns, B. Mysliwetz, and T. Christians, "An Integrated Spatio-Temporal Approach to Automated Visual Guidance of Autonomous Vehicles," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 20, no. 6, 1273-1284, Nov/Dec 1990.

[10] J.T. Fennema and O.W. Mitchell, "Vision Guided Servoing with Feature-Based Trajectory Generation," *IEEE Trans. Robotics and Automation*, vol. 5, no. 5, 691-699, Oct. 1989.

[11] C. Fennema, A. Hanson, E. Riseman, J.R. Bevride, and R. Kumar, "Model Directed Mobile Robot Navigation," *SMC*, vol. 20, no. 6, 1352-1369, Nov/Dec 1990.

[12] R.J. Firby, "Adaptive Exectution in Complex Dynamic Worlds," Ph.D. thesis, Yale University, 1989.

[13] M. Hebert, T. Kanade, E. Krotkov, and I.S. Kweon, "Terrain Mapping for a Roving Planetary Explorer," in *Proc. IEEE Robotics and Automation Conf.*, Scottsdale AZ, May 1989, pp. 997-1002.

[14] B.H. Krogh, "A Generalized Potential Field Approach to Obstacle Avoidance Control," in *Proc. Robotics Inter. Robotics Research Conv.*, Bethlehem PA, 1984.

[15] E. Krotkov, *Active Computer Vision by Cooperative Focus and Stereo*. Springer-Verlag, 1989.

[16] D. Marr, *Vision*. W. H. Freeman and Company, 1982.

[17] H.K. Nishihara, "Practical Real-Time Imaging Stereo Matcher," *Optical Engineering*, vol. 23, no. 5, 536-545, 1984.

[18] T.J. Olson and D.J. Coombs, "Real-Time Vergence Control for Binocular Robots," Tech. Rep., Computer Science, 348, June 1990.

[19] D.W. Payton, "Internalized Plans: A Representation for Action Resources," *Robotics and Automonmous Systems*, vol. 6, 89-103, 1990.

[20] L.G. Roberts, "Machine Perception of Three-Dimensional Solids," in *Optical and Electro-Optical Information Processing*, Cambridge MA: MIT Press, 1965, chap. 9, pp. 159-197.

[21] C. Thorpe and J. Gowdy, "Annotated Maps for Autonomous Land Vehicles," in *Proc. DARPA IUS Workshop*, Sep 1990, pp. 765-771.

[22] L.E. Weiss, A.C. Sanderson, and C.P. Neuman, "Dynamic Sensor-Based Control of Robots with Visual Feedback," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 5, 404-417, October 1987.

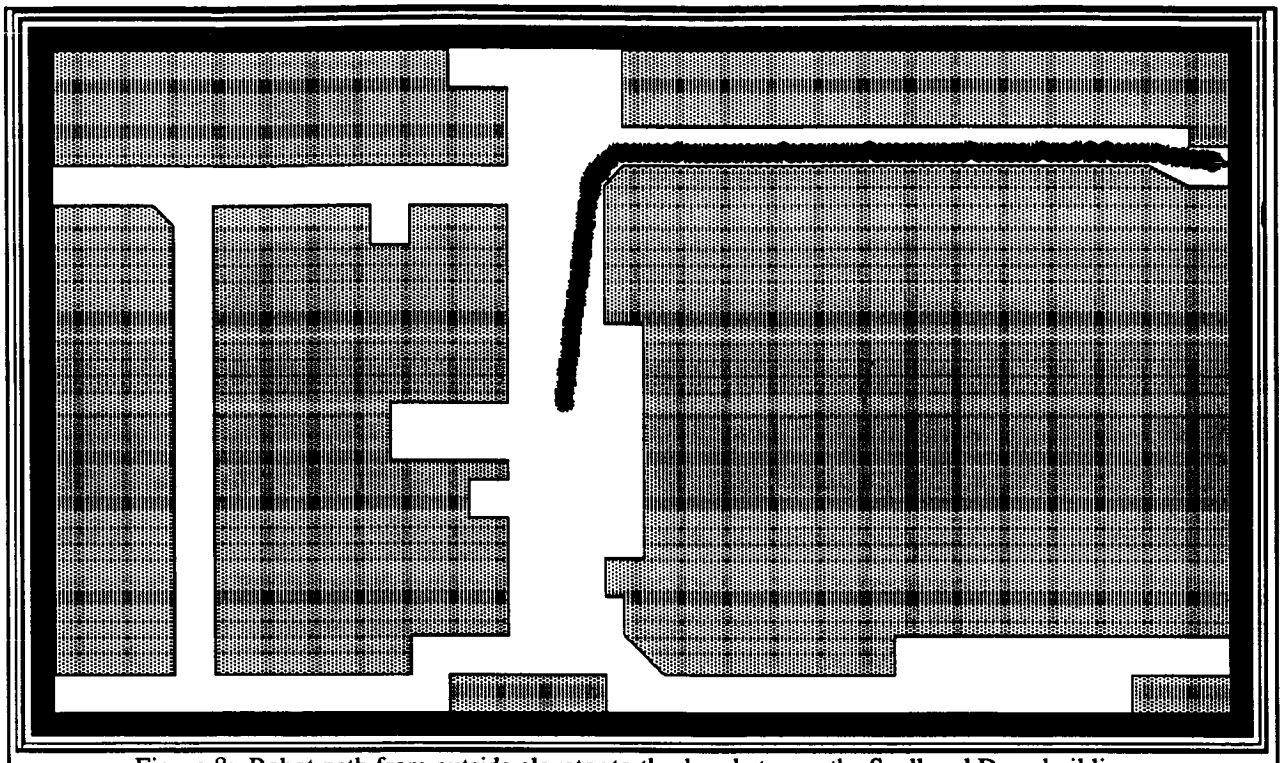


Figure 8: Robot path from outside elevator to the door between the Snell and Dana buildings.

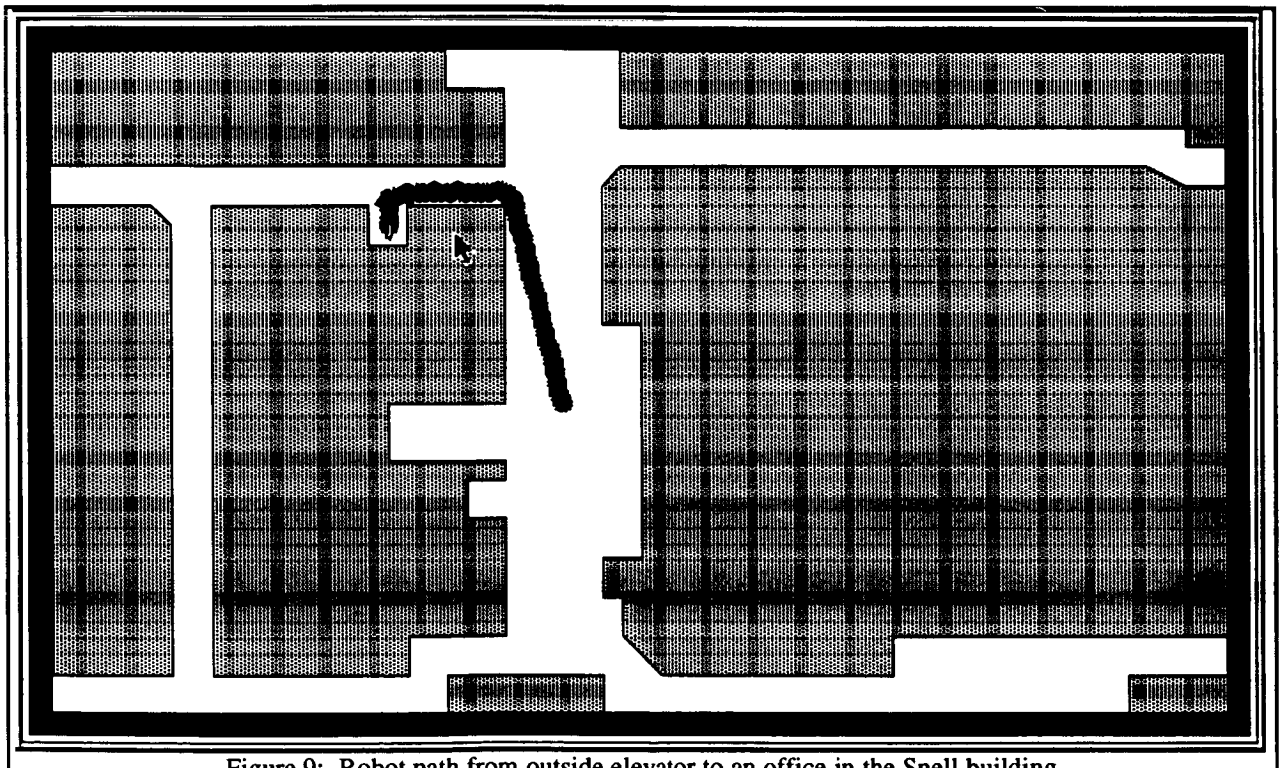


Figure 9: Robot path from outside elevator to an office in the Snell building.

A METHODOLOGY FOR THE GENERATION OF THE 2-D MAP FROM UNKNOWN NAVIGATION ENVIRONMENT BY TRAVELING A SHORT DISTANCE

N.Bourbakis and D.Sarkar
Binghamton University
T.J.Watson School
AAAI Lab
Binghamton, NY 13902

ABSTRACT

In this paper a methodology is presented for the generation of a 2-D map from an unknown navigation environment by traveling a short distance. The methodology proposed here is based on the synthesis of the knowledge extracted from consecutive free navigation spaces, during the movement of an autonomous mobile robot. The generation of the 2-D map of the space is classified into three cases: (a) space without obstacles; (b) space with standing obstacles; and (c) space with moving obstacles.

KEYWORDS: 2-D Map Generation; Navigation in Unknown Space; Synthesis of Space Segments; Knowledge Extraction from
Unknown Space.

This work is a part of RFG grant 1992-93

1. INTRODUCTION

Knowledge acquisition from an unknown navigation space is one the challenging problem facing in the modern robotics (intelligent robots) and AI today [1-8]. In realistic situations, only the boundary and the outer shape of the current free navigation space is known and the interior structure and/or formation of the space are totally unknown. The space that we are talking about, could be the surface of an unknown planet, destroyed battlefield, or demolished landscape. Depending on the situation, traveling through the navigation space by human may be hazandrous, expensive, time consuming, inefficient and most of all may be life threatening.

A technique for the generation of the 2-D space map was proposed by [3]. This method scans the entire area. When stationary obstacles exist inside the navigation space, this method goes around of each obstacle and at the end generates the complete 2-D sapce map. In case that the obstacles are moving in the navigation space this methodology does not work. Moreover, it is a time consuming approach, especially when the number of stationary objects is great.

In this paper we present a formal methodology, which extracts knowledge by traveling through an unknown navigation space and generates a complete 2-D map of the navigation environment. Little or no knowledge is assumed regarding the navigation space and knowledge is accumulated solely by traveling in it. The methodology of generation of the 2-D map has been studied under certain space conditions:

a. Space without any obstacles
b. Space with stationary obstacles
c. Space with moving objects

The methodology of generating the 2-D map is based on the graph modeling of the navigation space and the synthesis of the successive free navigation spaces. Moreover, the proposed methodology generates the 2-D map by traveling a short distance in the total space.

This paper is organized into six section. Section 2 provides some definitions and notations. Section 3 deals with the representation of the knowledge extracted from the navigation space. Section 4 presents the synthesis of the shape-graphs. Section 5 discusses the generation of the space map and section 6 concluded the overall presentation.

2. NOTATIONS AND DEFINITIONS

In this section we provide a number of notations and definitions in order to accommodate the understanding of the following sections.

Notation 1: GNS represents the global navigation space.

Definition 1: A 2-D obstacle, b(j), jeZ.is defined as the two dimensional surface, Sb(j) $\subset$ GNS, where a moving object (robot R) cannot go through it, and the visual and laser rays stop or reflect on it.

Definition 2: A robot, R(n), neZ, is a moving "obstacle" in GNS by using its own power to do so.

Notation 2: BS represents the set of all the obstacles in GNS,
BS C GNS.

Notation 3: RS represents the set of all the robots in GNS, RS $\subset$ GNS.

Copyright c 1994 by N. G. Bourbakis and D. Sarkar. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

Notation 4: Sr(n) represents the 2-D surface covered by a robot R(n) in GNS.

Notation 5: E(b) = $\bigcup$ {Sb(j)} and E(r) = {Sr(n)} represent the total surfaces covered by the total number of obstacles and the total number of robots in GNS respectively.

Definition 3: The total free navigation space inside GNS, is defined as FNS = (GNS - [E(b) + E(r)]).

Notation 7: t(i), ieZ, represents the current time.

Notation 8: NS(t(i)) represents the navigation space at the time t(i).

Definition 4: The current free navigation space $FNS^n(t(i))$ is defined as the free space perceived by the robot R(n) at the time t(i), $FNS^n(t(i)) \subset FNS$.

3. KNOWLEDGE EXTRACTION AND REPRESENTATION FROM THE FREE NAVIGATION SPACE

In this section the extraction and represention of knowledge from the geometric form of the current free navigation space $FNS^n(t(i))$ perceived by a moving robot R(n) are presented. In particular, the extraction is related with the construction of the shape SH ($FNS^n(t(i))$ of the current free space. Figure 1 shows graphically the generation of the shape. Then, the relationships among the straight line and curve line segments of the shape $SH^n(t(i))$ are defined and the representation of the shape (knowledge) with the use of syntactic and semantic information is obtained by using directed graph with attributes [6].

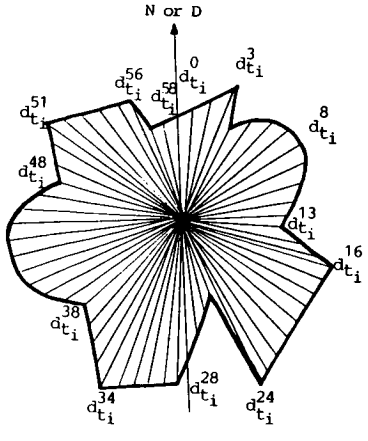


Figure 1: a) Construction of the FNS shape

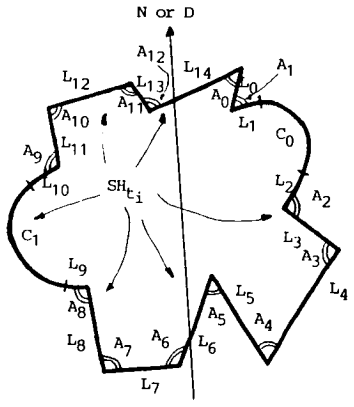


Figure 1: b) Extraction of the FNS shape

4. SYNTHESIS OF CONSECUTIVE SHAPE-GRAPHS

In this section, we describe the synthesis of successive shapes expressed in graph forms [5].

Two shapes $SH^n(t(i))$, $SH^n(t(j))$, with i#j can be considered for synthesis if the graph form of these shapes satisfy the following basic proposition:

Proposition : Two shapes $SH^n(t(i))$, $SH^n(t(j))$, i#j, extracted by the same robot R(n) at two consecutive time intervals in the same navigation space, can be considered as candidates for composition into a new shape $SH^n(t(ij))$ if and only if their graph forms have at least one common node, $G^n(t(i)) \rightarrow N(k) = N(m) \in G^n(t(j))$ with the same properties Pr and the similar relationships Rs with the other nodes, where Pr={size,color,length,curvature,etc}, and Rs={connectivity, parallelism, symmetry, relative-distance, relative-magnitude,etc}.

Starting the synthesis process, we have to search initially the graph form of each shape for graph-nodes that satisfy the proposition above. If we detect such a node in the first graph, then we save its characteristics and we proceed with the nodes of the second candidate graph. If there is at least such a node in the second graph, then we attempt to match its characteristics with the corresponding ones of the node, which belongs in the first graph. If there is a successful matching, then these particular nodes will be the starting point for the synthesis of the two graphs. More specifically, the synthesis process of two consecutive shapes (using their graph-forms) is based especially on the connectivity relationship (angle) of the nodes with the same properties. Figure 2 shows graphically, the synhesis of straight line segments where the segment with the maximum clockwise angle is eliminated. The synthesis process of the rest nodes for these two candidate graphs is based on the detection of closed subgraphs and determination of their "extended" new subgraph forms.

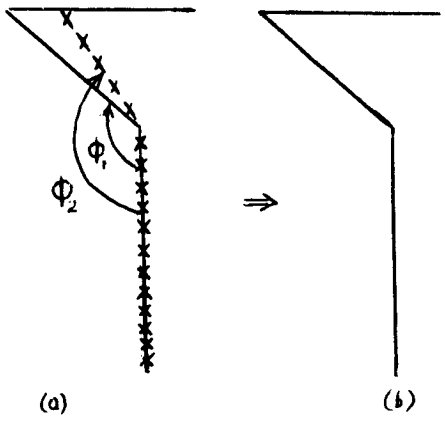


Figure 2: Synthesis of staight line segments taken from two consecutive shapes.
a) Matching of the segments
b) The new segments after synthesis

At the end of the synthesis process, the two shapes generated a new shape which represents the map of two consecitive free navigation spaces. By repiting this synthesis process, finally the map of the navigation space will be produced. Noe there is a critical question. For how long does a robot have to travel in order to generate a complete 2-D map of an unknwon space? The next section attepts to give some answers to the question above.

5. GENERATION OF THE SPACE MAP BY TRAVELING A SHORT DISTANCE

The generation of the space map requires the classification of the unknown space into three main categories:

. *Space without Obstacles*
. *Space with stationary obstacles*
. *Space with moving obstacles*

5.1. SPACE WITHOUT OBSTACLES

Here the problem is to generate the 2-D map of the navigation space by traveling a short distance, under the condition that no obstacles exist inside the space. The solution is rather trivial if the shape of the space is regular geometric one, as shown in figure 3. In this case, the shape of the space is simple and the center of gravity (or geometric center) Cgis ease to be located. Thus, if a robot detects such a shape then it calculates the center of gravity. This means that the robot has to travel a distance d[p(x,y),Cg] from its current location p(x,y) to Cg. When the robot will reach the Cg then its confidence function takes its maximum value, thus the 2-D map can be generated.

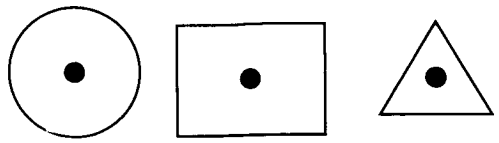


Figure 3: Some primitive geometric shapes

In case however where the shape of the navigation space is not a regular geometric shape, the problem becomes more complex, see examples in figure 4. For these cases, we partition the shape of the navigation space in order to obtain a set of simple (primitive) geometric regular shapes, figure 5. Thus, for each primitive we define a center of gravity. At this point, all the centers of gravity are connected by straight line segments (if possible) and the robot has to reach the nearest center of gravity. From that center of gravity the robot will travel on the line segments (gravity-line), which connect the centers of gravity, by minimizing the traveling distance for the generation of the 2-D space map, see figure 6. Note that the confidence function takes its maximum value by traveling on the gravity-line. It is also important to be noticed that if the

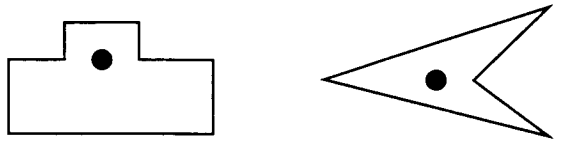


Figure 4: Complex geometric shapes

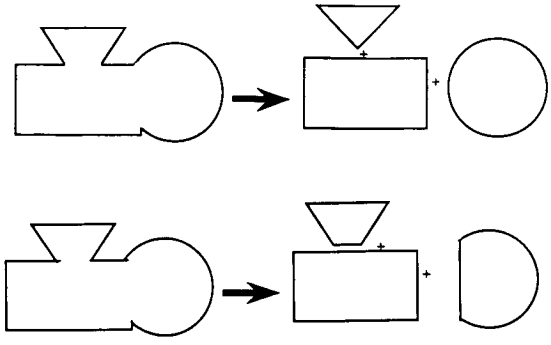


Figure 5: Partitioning of a geometric shape into a set of primitive shapes.

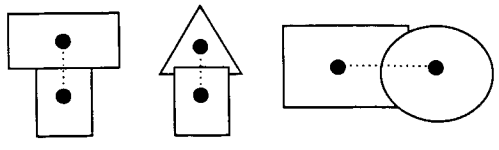


Figure 6: Connection of the centers of gravity.

navigation space is partitioned into "n" regular geometric shapes and if all the centers of gravity are connected within the space, then visiting these Cg points in an optimal way is equivalent to finding a permutation g1,g2,g3,...gn, which minimizes the total traveling distance (traveling salesman problem). A heuristic solution is proposed here in order to avoid such an NP complete problem. The solution is coming by generating the "skeleton" (thinning) of the navigation space [9], see figure 7. Thus, the complexity of the problem is reduced by eliminating the partitioning process and the complexity of the connection of the centers of gravity. In this case, the robot has to travel on the skeleton line in order to generate the space map.

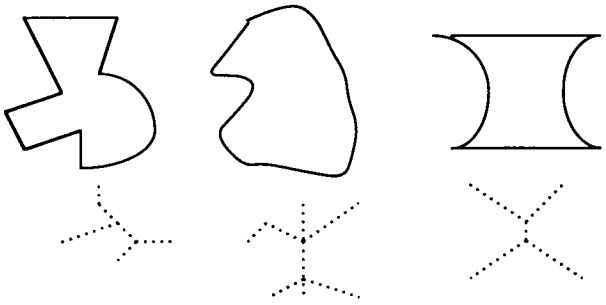


Figure 7: Inregular shapes and their skeletons

5.2. SPACE WITH STATIONARY OBSTACLES

The solution to this particular problem is similar to the generation of the skeleton line. Then the skeleton line is converted into an equivalent graph, see figure 8. The generation of the complete graph is based on the synthesis of the current graphs generated by the movements of the robot. Firtsly, the robot uses its starting point as the "root" of the current graph G1 (see a1 in figure 8). Thus each segment of the current skeleton represents a branch of the graph. If the robot will be moved to a3 point, then a new graph G2 is generated and the new graph is synthesized with the pervious one for the production of a graph G3 which represents two consecutive free navigation spaces.

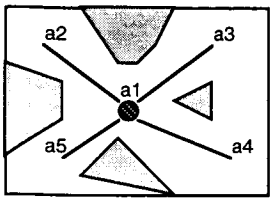
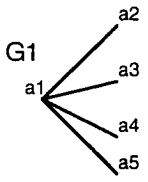


Fig. 1



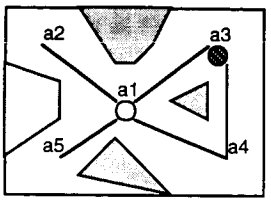


Fig. 3

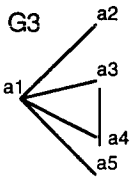


Fig. 4

Figure 8: Graphical representation of a space with stationary objects, and the graph representation of the free space.

The important feature of the G3 graph is that it has the ability to detect some graph nodes, which were perceived from the previous position. Such a case is the point a4. Thus this important observation plays a very significant role for the shorter distance to be traveled. More specifically, the robot has information related with the point a1, a3 and a4, thus there is no need for it to travel on the segment (a1a4), since it "knows" some information about the shape and the path related to a1a4 brach of the graph. Another interesting point is the generation of intersected nodes. This means that, as the robot is moving and the new graph is generated, there are some locations from which the graph generates new braches, which intersect other braches constrcuted previously. The reason is that the robot can see areas viewed before from different angle. Thus it combines that knowledge for the generation of the new graph.

5.3. SPACE WITH MOVING OBSTACLES

In this case, which is also the most complex, the methodology followed is similar with the skeletonization of the navigation space with the only difference that the moving objects inside the space request one extra processing step. This step is the real-time detection of the moving objects in the each current free navigation space [6] and the appropriate reconstruction of the new graph.
The complexity of this case increases significantly when the number of moving objects in the navigation area increases too.

6. CONCLUSIONS

In this paper, a methodology for the generation of the 2-D space map by traveling a short distance was presented. More specifically, the study of the problem was partitioned into three subcases, 1) space without obstacles; 2) space with stationary obstacles; and 3) space with moving obstacles. The advantage of this methodology is the ability to minimize the redundancy during the traveling and maximize the confidence function for the generation of the 2-D map. The main disadvantage of the methodology proposed in this paper is the risk of generating a 2-D space map with some lost of information.

REFERENCES

[1] T.Lozano-Perez,Spatial Planning: A configuration space approach, IEEE Trans. on Computers, 32,2,1983, pp.108-120.

[2] R.Brooks,Solving the find-path problem by good representation of the free space,IEEE Trans. on SMC, 13,3,1983,pp.190-197.

[3] J.Barraquand and J.Latombe,Robot Motion Planning:A distributed representation approach,IEEE J. on Automation & Robotics, 1991

[4] N.Bourbakis, Real-time path planning autonomous robots in a 2-D unkonwon space,IJ.on Intelligence & Robotic Systems 4,1991,pp.333-362.

[5] N.Bourbakis, Knowledge-based acquisition during real-time path planning in unknown space, in Learning and Planning Methods and Applications, WSP,311-331,1991

[6] N.Bourbakis, An knowledge acquisition scheme using attributed graphs,Technical Report 1989.

[7] L.Gonzenes,Collision avoidance for robots in an environmental flexible assembly cell,Proc. IEEE Conf. on Robotics,CA,1984.

[8] M.Maas, A.Mogzadeh and N.Bourbakis, Fusion of image and laser sensory data for 3-D modeling of the navigation space, Int. AIAA Conf. on Intelligent Robots, March 1994, TX

[9] N.Bourbakis N.Stefenssen and B.Saha, A parallel skeletonization methodology and its VLSI implementation, TR 1993, sub. to Int. Journal.

AIAA-94-1252-CP

SIMPLIFYING APPLICATIONS SOFTWARE FOR VISION GUIDED ROBOT IMPLEMENTATION

Charlie Duncheon
Adept Technology, Inc.
San Jose, California

Abstract

This paper begins with the background on how robotic implementation has flourished in Japan while only moderately growing in United States manufacturing. Contributing reasons are provided with focus on the constraint of the time and difficulty of robot applications software. Particular focus is made on the largest robot application segments in the world markets, material handling and assembly. Manufacturing demands of the 1990's are cited and scenarios of 21st century plants are described. Successful implementation of vision guided robots for these plants is described by use of the following system building blocks:

1. An industry standard form factor, open architecture hardware control platform.
2. A proven and integrated real time operating system and factory automation language within the platform that provides direct support for sophisticated motion control and real time sensing.
3. A building block, icon/menu based application software approach for both developing applications and for easy factory floor interface.
4. An open architecture platform that allows other proven operating systems and/or cell control hardware/software solutions to coexist on the same back plane with the core motion/vision operating system.

Background

Despite robot technology being invented in the United States, the implementation rate in the US market fell behind that of the Japanese by tens of thousands of robots per year by the end of the 80's. The trend continued in 1993 despite a new parity in the cost of capital between US and Japan. New "Agile Manufacturing" thrusts have emerged in US manufacturing yet they are noticeably short of true flexible automation strategies. In the early 80's Prudential Bache projected a US robot business of $2 billion by 1990. The market actually reached $500 million in the early 90's. The current installed base of industrial robots in Japan is 400,000 units, while in the US it is only 50,000 units. Japan installs more robots per year than the total installed base in the US.[1]

Why the slow adaptation of industrial robots and flexible automation in the US compared to the continuing robust growth in Japan? First of all, Prudential Bache was correct in identifying $2 billion of robot applications that should have been implemented in the US. In fact, it would have been even more had the technology been implemented at the Japanese rate. The fundamental constraint to growth of the robot industry was the difficulty in implementing the technology. Suppliers overestimated the time, capacity and technical skill level manufacturers had to implement automation. Financial justification followed the same parameters as applied to traditional hard automation, leading to only the most challenging applications gaining financial approval. This only compounded the problem of successfully implementing the technology in manufacturing.

A study by Adept Technology, Inc.[2] showed that in the 1980's the cost to design, write, debug and document application software was as high as 50% of the total robot system costs. And for the most part the software developed was custom to each application. Application software development, debug, and implementation was also determined to be a common bottleneck to system implementation schedules. Other robot companies and robot systems integrators have come to the same conclusions and have taken steps to standardize application software. In 1992 and first half 1993 statistics released by the Robotics

Copyright © 1993 American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

Industries Association show healthy increases in the implementation of robots in the US market, due in part to more enabling software. 1992 orders for US based robot manufacturers grew 21.5%, and first half 1993 orders jumped 40%.[1]

This paper will discuss how the utilization of **existing** and **proven** hardware and software modules which minimize custom software development cannot only increase US robot implementation beyond the current improved rates, but put the US in the global lead. There will be no formulas or equations; manufacturing engineers who are our vital hope for competitiveness have no time for such. They have plenty of their own process algorithms to address. They care about practical suggestions that allow them to address the cost and quality issues without compromising lead time to market. That is what this paper will attempt to address.

Manufacturing Forces of the 90's

There are many forces influencing manufacturers today but three main forces stand out that will determine global manufacturing strategies:

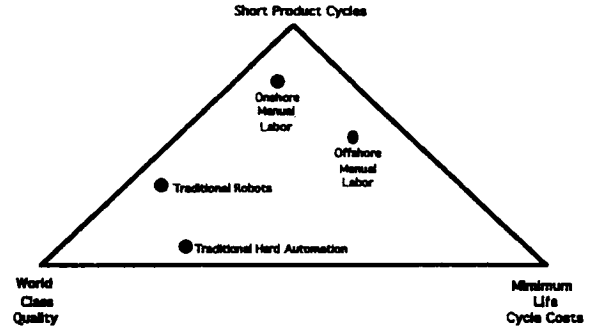
1. World Class Quality - World class quality is not an option but the price of admission for those manufacturers who want to survive the decade. The "Six Sigma" commitment by Motorola is one of the more publicized commitments to this imperative.

2. Minimum Life Cycle Costs - The 90's is being dubbed as the "value decade", as manufacturers constantly evaluate their manufacturing costs against global competitors. As governments realign the world markets via NAFTA, GATT and EC agreements, even those manufacturers who limit themselves to domestic markets will see global competition in their home markets.

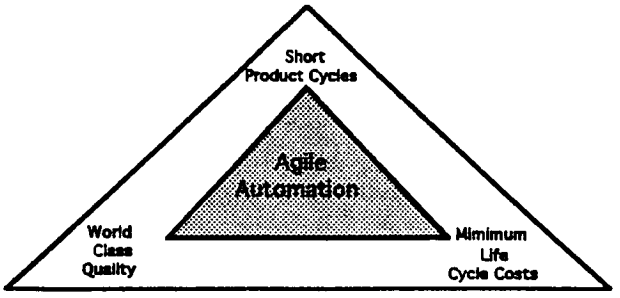
3. Minimum Product to Market Times - The laptop computer used to write this paper is currently nine months old and has already been obsoleted by a higher performing model.

Conventional high volume, low mix production strategies are in conflict with this force. Manufacturers are being challenged to have shorter changeover times or running multiple products simultaneously on the same line. One power supply manufacturer has an objective that a customer can specify a custom power supply with a lot size of one and receive it within one day, built with flexible automation.

Unfortunately, the above three forces have traditionally been in conflict with each other. Automation might have addressed quality forces but did not support rapid lead time to markets. Traditional robot automation may not have met cost objectives. And many US companies found that offshore labor did not meet quality requirements.



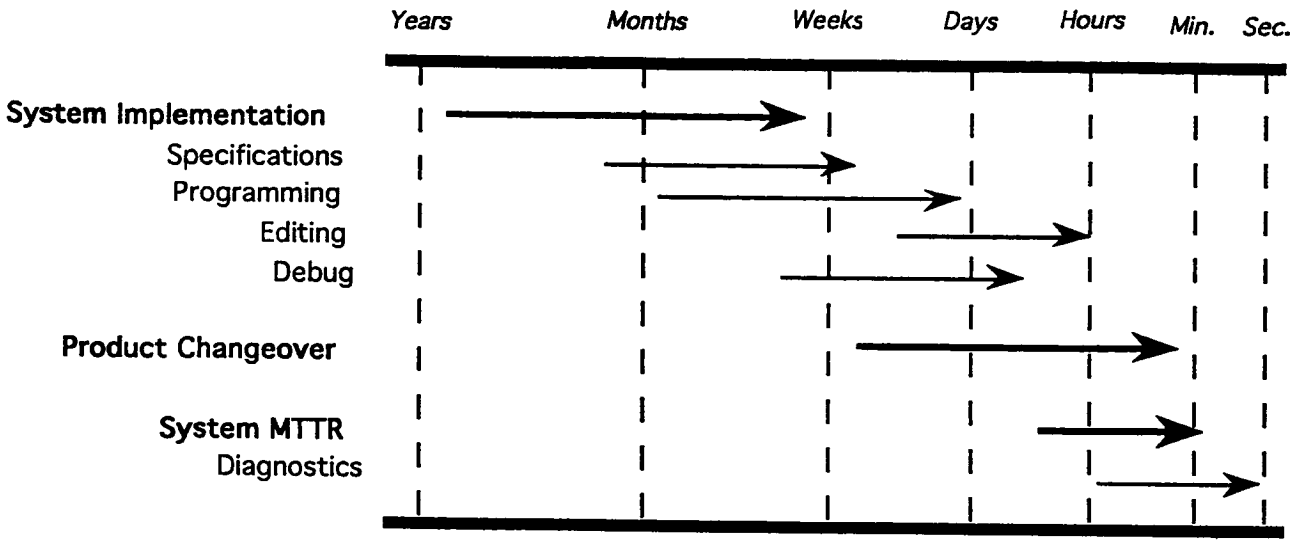
The RIA and its member companies believe that there does not have to be a tradeoff among these forces. The belief is that properly applied, "Agile Automation" can "shrink' the triangle of tradeoffs.



Adept Technology, Inc has developed a long range strategy for minimizing flexible automation cell implementation and changeover time with the development of vision guided flexible feeding technologies and modular functional and application software. These

Rapid Deployment Automation

Objective: Conservation of Time



efforts are part of Adept's overall vision of "Rapid Deployment Automation".

Rapid Deployment Automation is an overall strategy focused at reducing the time, and thereby the cost, required to implement flexible automation. Given that systems typically cost 2x the direct hardware costs, Adept feels strongly that reducing engineering content will be a primary driver to reducing the cost of flexible automation. Rapid Deployment Automation also address directly the product changeover / time to market issues discussed above.

Flexible feeding is a key element in Rapid Deployment Automation, as by nature it reduces the amount of hard tooling in a robot cell, which reduces costs and promotes rapid development of cells.[3] Flexible feeding replaces traditional part specific feeding systems such as bowl feeders and precision dunnage with intelligent sensor based robotic application software. Sensing is provided by integrated machine vision and real time force sensing, which are used to locate random or loosely oriented parts on simple conveying or infeed systems. Flexible Feeding software includes modules and algorithm's to locate the parts, determine their orientation, control the conveyors and recirculating devices, and acquire and place the parts. The Flexible Feeding concept, and the specific software modules that drive it, are examples of how software can dramatically simplify the overall engineering task in a flexible automation cell.

21st Century Factories

If Adept and other RIA companies are successful, the following scenario is possible for the year 2000 manufacturing assembly plants:

1. Small, nimble plants, less than 250 employees, located close to consumer bases of population or major OEM customers.

2. Lot size of one, same day delivery to customers. Rapid changeover from one part to another and rapid implementation of incremental process and component part improvements and changes.

3. Vision guided robot and flexible feeder based production. Limited investment in hard tooling. Intelligent software based feeding and fixturing.

4. No software language based programming within factories. Simplified set up of highly intelligent software modules with imbedded process knowledge by floor personnel.

How Do We Meet the Challenge?

One way we will not meet the challenge is to continue to develop application software and/or motion control platforms "from scratch" because of what appears to be unique or process specific requirements. The development cycles for applications will exceed the product cycles themselves. The approach certain robot and systems suppliers are taking is to provide function and application software modules that serve as building blocks to the final line application program.

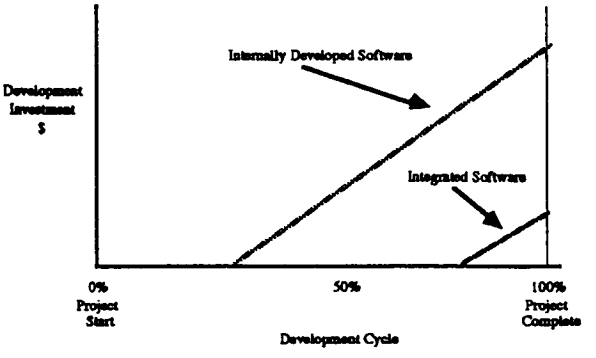
This modular approach to software development is not new. There are countless libraries today of software functions from various hardware and software vendors. The burden in using modular libraries is in the linkage between modules. This fine tuning still requires highly skilled programmers, and significant time. The new approach is to provide these modules within a structure or framework that defines how the modules work together. Unlike object oriented programming, which offers a similar strategy, this new generation of factory automation software includes high level interfaces and process knowledge about the tasks themselves to allow setup by non programmers.[4]

The best way to describe such an approach is with an application. Assume we have a dental adhesive applicator product with a cylindrical body with one open end, an impeller that must insert into the body, and a cap that must be inserted at the top of the body and impeller. The body arrives open end up in multiples on a tray moving on a roller conveyor and stops against an activated fixture. The dunnage that carries the body is general purpose, and does not provide significant precision in locating the bodies. The manufacturer is dealing with a product with a short life cycle and wants to quickly reuse the robot, grippers, feeders, and software when any of the three parts change in design. A single vision guided flexible feeder for the impellers and caps, and vision guided registration of the bodies in the trays along with inspection for acceptable inner body diameters are among the cell design parameters.

The long path to a solution would be to interface one supplier's vision system and language with a second motion system and language, with custom software written in C, perhaps on a PC platform. Not only is the development cycle long, but history has shown debug activity for this approach to extend well past floor implementation.

This potentially ill fated path relates to a lack of empathy of many developers with regard to "real time" in a robot cell on the factory floor vs. that in the work station environment. Deterministic, multitasking operating systems that can do context switching in micro seconds are crucial when expensive grippers with expensive parts are on a collision path. More important than part and gripper damage, system downtime is unaffordable. Manufacturers like Toyota and Michelin are requiring 40,000 hour MTBF performance from equipment suppliers. In other words, bugs or errors at the work station are manageable, but unacceptable on the assembly floor.

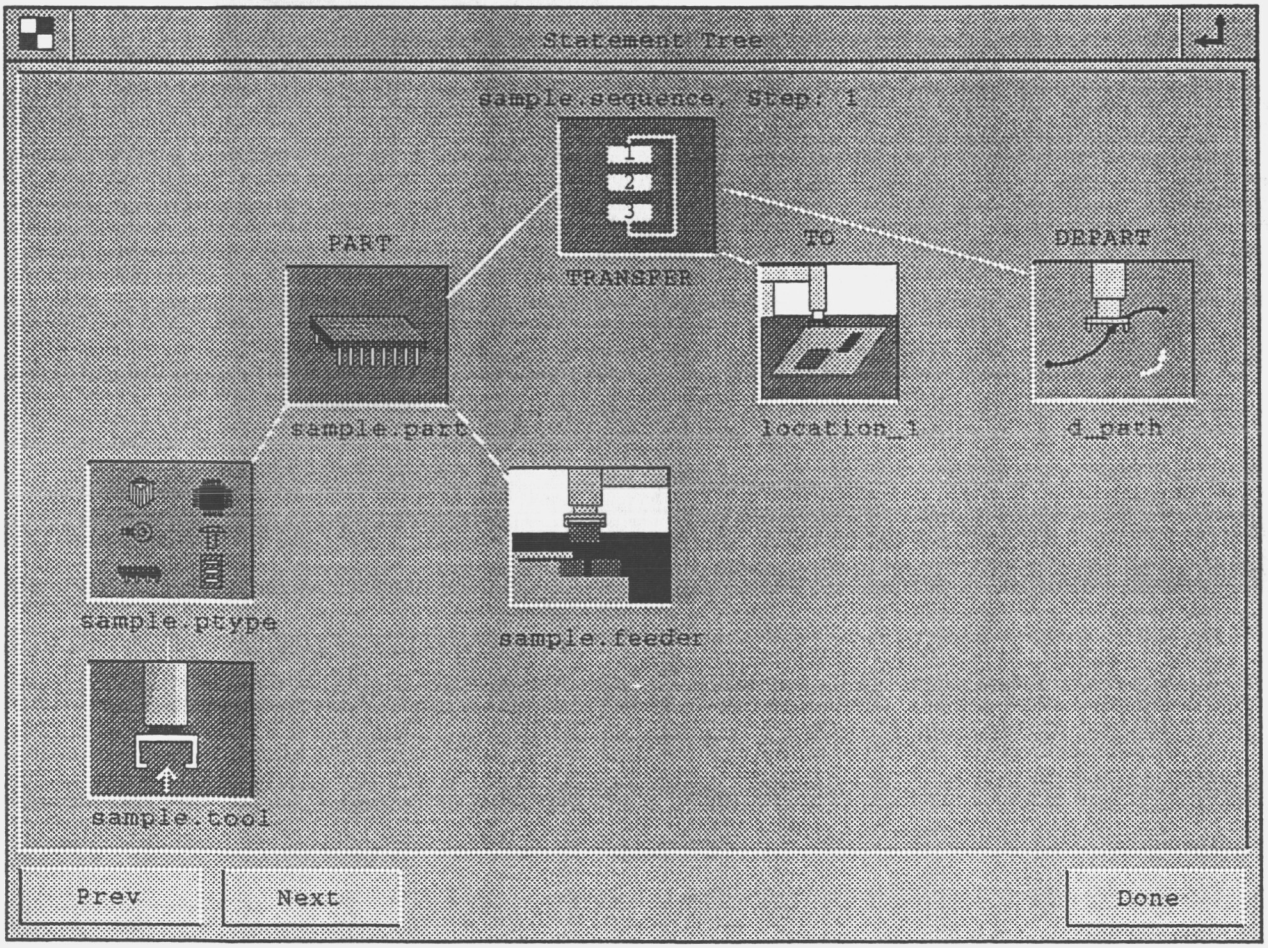
The integrated control platform offers a much lower cost, schedule time, and risk approach to this solution. A platform that has thousands successfully running applications while keeping motion, vision and force sensing, and communications software under one language is the optimum place from which to start for this application.[5]

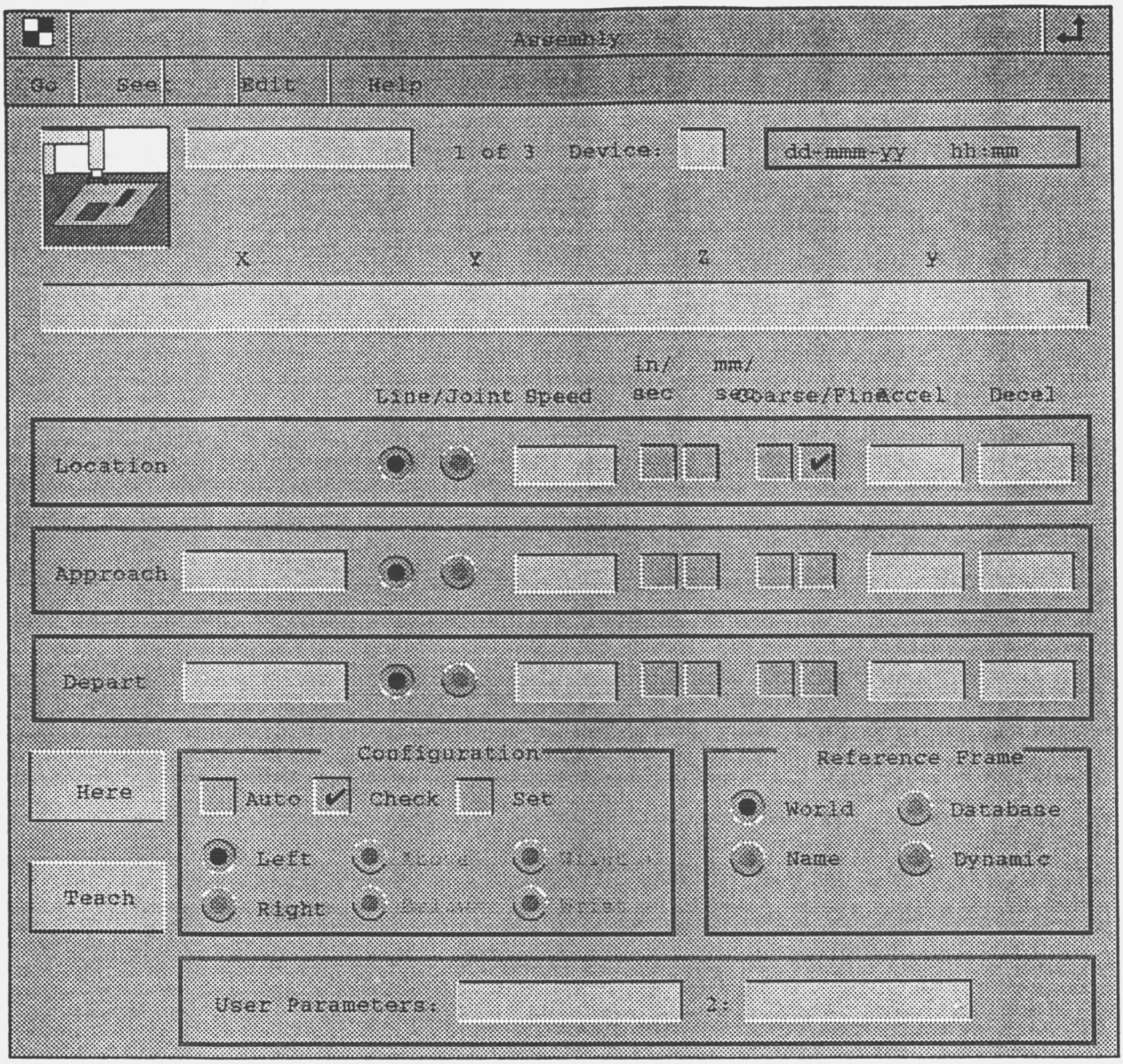


An Adept control platform that control's Adept robots or any other robot mechanism would be made up of "wrung out"functional software modules that build into an overall applications module. These functional modules are menu

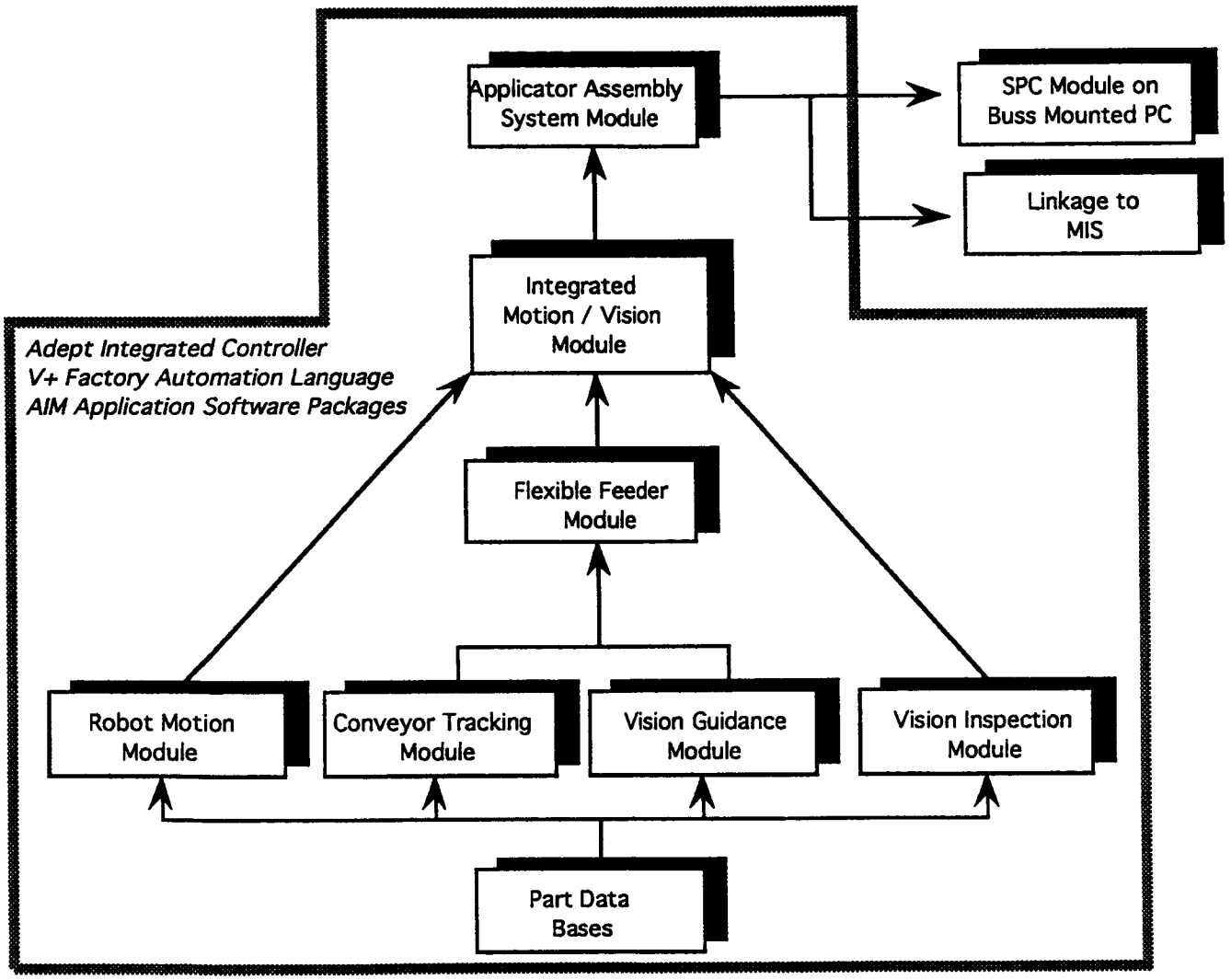
driven and utilize a common data base structure and common global variables. Modules are tied together to create the specific application module, imbedding any necessary process knowledge, by use of the high level V+ programming language. Adept's standard functional modules and tools for vision and force guided assembly are represented by icons.

Adept's patented AIM [tm] (Assembly Information Manager) environment allows the developer to create additional linked icons as required to represent combinations of Adept modules and any application customization or process imbedding.

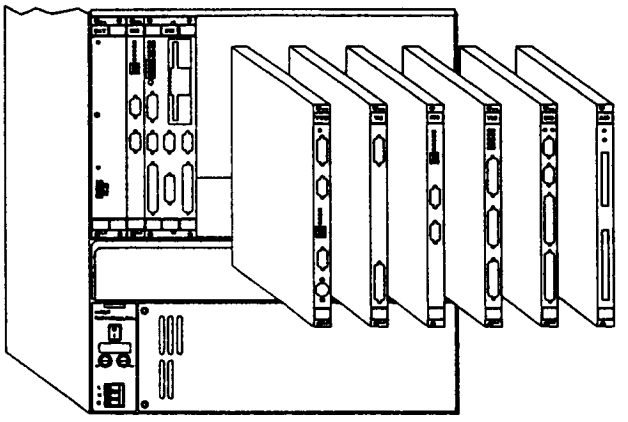



Assembly
Go
See
Edit
Help
1 of 3
Device:
dd-mmm-yy
hh:mm
X
Y
Z
Y
Line/Joint Speed
in/ sec
mm/ sec
Coarse/Fine
Accel
Decel
Location
Approach
Depart
Here
Teach
Configuration
Auto
Check
Set
Left
Right
Reference Frame
World
Database
Name
Dynamic
User Parameters:
2:

Example of Robot Cell Software Structure



So the impeller assembly application can be quickly developed with bug free software modules which also leave an rational user interface for modifying on the factory floor. Now let's assume the manufacturer has strict FDA requirements for Statistical Process Control records and has invested heavily in a PC based SPC program that he wants to apply to any workstation. The Adept MV controller contains a standard VME back plane with open slots for other boards. The manufacturer can have a VME board with an imbedded PC inserted on the Adept back plane and communicate as required with the Adept operating system and AIM. Customization is limited to defining what variables are needed by the SPC module.



Summary

The approach described here is simple: Use commercially available software and hardware wherever possible to minimize system costs, schedules, and risks. There is an abundance of software and hardware technology that does not have to be reinvented. This approach is what put the Japanese in a leadership role in the 80's. The US now can do the same with a much more flexible array of vision guided robot technologies.

1. Robotics Industries Association, Ann Arbor, Michigan.
2. Adept Technology internal integration study. November, 1989.
3. G. Orelind, "Flexible Part Feeding for Robotic Workcells", Robotics World, Summer 1993.
4. J. Campbell, P. Cencik, "General Purpose Scalable Motion / Vision Controllers", Proceedings, ISTA Conference on Lean Manufacturing, August 1993.
5. J. Campbell, "The Economics of Utilizing an Integrated Control Platform for OEM Motion Control", Motion Control, February 1993.

AIAA-94-1253-CP

AN OPEN ARCHITECTURE MOTION CONTROLLER

Lothar Rossol
Trellis Software & Controls, Inc.
Rochester Hills, Michigan
AIAA/NASA Conference on Intelligent Robots
In Field, Factory, Service, and Space
March 1994

Abstract

Commercial controllers for robots are typically custom-designed with closed architectures on proprietary hardware and software platforms. However, the cost of *open controllers* that use standard computer hardware and software platforms is rapidly decreasing. It is now practical to build an open controller for sophisticated robot and general motion control using off-the-shelf components. Such open controller designs allow the user to standardize on hardware platforms such as VME, and on operating systems and user interfaces, such as UNIX or Windows. This paper describes *Nomad*, an open architecture motion controller. Nomad consists of a set of software modules designed to control robots, various specialty machines, and machine tools. The base operating system for Nomad is LynxOS, a POSIX-compliant real-time UNIX system. LynxOS, and hence Nomad, runs on a number of hardware platforms, including PC-ATs, VME-based PCs, Motorola and RISC processors. Nomad provides for sensor-controlled robotic motions, with user replaceable kinematics. It is programmable in C, with full UNIX compatibility, including X Windows and MOTIF. Specialized programming interfaces and languages have been added. Open architecture controllers, as represented by Nomad, will have a major impact on the robot control industry.

Introduction

Typically, motion controllers for robots and other machines have been developed based on closed architectures and proprietary platforms. The hardware, the operating system, and the software were custom designed. The result was a closed system that was inflexible and expensive to develop and maintain. In addition, closed architecture controllers could not take advantage of the rapid improvements in cost and performance driven by high volume markets outside the motion control industry. Also, standardization on common platforms by customers and end users was impossible.

In the past, the advantages of such custom architectures have been cost and performance. However, the cost of off-the-shelf standard computer hardware has dropped dramatically and performance has increased substantially in the recent past. The result of these rapid advances is that it has now become feasible to build sophisticated robot and general motion controllers using off-the-shelf components. These compare favorably in price and performance to custom designs.

If standard real-time operating systems are also used in their designs, then open architecture controllers will automatically leverage future advances in hardware, driven by R&D funding in high volume consumer markets. That is, such open designs will become more and more attractive in price and performance over time.

In addition to attractive cost and performance, open controller designs allow the customer to standardize throughout his factory on platforms such as VME, and on operating systems and user interfaces, such as UNIX or Windows. This is an attractive advantage not possible with the various custom designs. Also, such controllers can be tailored exactly to the requirements of the customer's application and the machine being controlled. This is expensive or impossible with closed custom architectures.

Published in similar form in *Proceedings*, International Robots and Vision Automation Show and Conference, April 1993, Detroit, MI.
Copyright © 1993 by Trellis Software and Controls, Inc. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

With Nomad, Trellis Software & Controls, Inc. is introducing open architecture software for motion control. Nomad software modules can be combined with standard off-the-shelf computer hardware and software to build robot or other motion controllers that are standardized and yet tailorable exactly to specific applications and machines.

Nomad Overview

Nomad was designed to be the foundation for open architecture motion controllers for robots, various specialty machines, and machine tools.

The base operating system for Nomad is LynxOS, a POSIX-compliant real-time UNIX system. LynxOS, and hence Nomad, runs on a number of hardware platforms, including PC-ATs, VME-based PCs, Motorola and RISC processors. Nomad-based controllers will automatically benefit from future advances in functionality and pricing of these hardware platforms.

A controller built with Nomad software modules and off-the-shelf standard computer components provides full power of sensor-controlled robotic motions, with user replaceable kinematics. It is programmable in C, with full UNIX compatibility, including XWindows and MOTIF. Specialized programming interfaces or languages can also be added.

Nomad-based motion controllers can be uniquely tailored to specific machines and applications. Through the power of C, X Windows, and MOTIF, tailored application packages can be developed, resulting in controllers that are very simple to use from the end user's perspective. These application packages can be menu-based or teach pendant programmable or whatever is appropriate in each instance. With application-specific interfaces, the end user need not be faced with having to learn the intricacies of an operating system or a programming language.

Nomad Components

Nomad components include *TMOS*, a Cartesian trajectory generator that provides the full power of sensor-controlled motions and a variety of industrial servo and I/O interfaces. The trajectory generator allows for six degree-of-freedom Cartesian position offsets in real time and coordination with multiple auxiliary axes. Kinematic solutions for different machine configurations can be incorporated into TMOS. The I/O control provides precise synchronization of both discrete digital and analog I/O with motion.

C-WORKS, the second component of Nomad, provides the user environment for Nomad. It consists of a C library for communicating with TMOS, a system configuration tool for Nomad, control panel functions, and a set of demonstration programs.

A wide variety of optional *Utilities* represent the third Nomad component. These utilities include a graphical servo tuning tool, a graphical machine simulator, a command line interface to Nomad, teach pendant support, and others.

Nomad was designed to be open and modular. This allows Trellis to provide additional combinations of user environments with other motion systems in the future. For example, other user environments planned consist of a robot language and an NC environment for Nomad.

The remainder of this paper will cover TMOS, the Nomad trajectory generator.

TMOS Interface to Nomad

This section describes the *High Level C Library (HLCL)* interface to TMOS. The HLCL provides many conversion functions for various forms of user data, performs integrity checks on user supplied data, provides parameter schedules to simplify motion programming, and hides internal TMOS data structures from the C program, minimizing the need for program modifications with TMOS product enhancements.

In addition to advanced motion commands, the HLCL interface to TMOS contains a number of unique features that allow programmers to communicate position information in many different formats, use "schedules" to reduce complexity of simple applications, generate

complex trajectories with minimum effort, program sensor-guided motions, generate off-line paths, and provide user control of error handling.

Connecting to TMOS

Figure 1 illustrates the interaction between a C program using the HLCL and TMOS.

Some HLCL function calls work synchronously with TMOS by sending it a message and waiting for a response before returning to the user program. Some functions, such as a request to set a digital output, require no response and return immediately after sending the message. Finally, some functions, such as status requests, interact with an information base that TMOS updates.

Motions generally execute in parallel with the user program. That is, HLCL calls that initiate motion will return when the motion is queued. When motions complete, a message is sent back to the user process. A user definable *callback* function is then used to process the completion.

Multiple user programs may *login* to TMOS. Since the information described above is kept within the space of each user process, each process has its own *context* and will not interfere with other programs that might use or modify the same parameters.

Position Allocation and Data Types

Languages for motion control typically provide one or more specific data types for position data which can be declared and allocated within a program. These types must then be supported in communications with the outside world through files or networks. The inevitable problem is one of compatibility with off-line generated data or with new and improved releases of software.

The HLCL for TMOS allocates position data internally in an undocumented format called a *TMOSPos*. HLCL calls can be used to convert between a variety of user formats and a TMOSPos. TMOS will then return a *handle* to the TMOSPos for later use in motion and other calls. The burden of storage and communication of the user's data is left with the user.

For example, an *Euler* format for position data can be declared and allocated in the user's C program. He can store and retrieve such data using files or the network. He would then call a TMOS routine to convert that data to a TMOSPos and return a handle to the user. Later, the user's C program can issue a motion call to TMOS using the handle.

In this way, issues of upward compatibility with future versions of TMOS are avoided, and the user is free to archive his data in whatever format and precision he chooses. (That is, the format of a TMOSPos is not intended to be

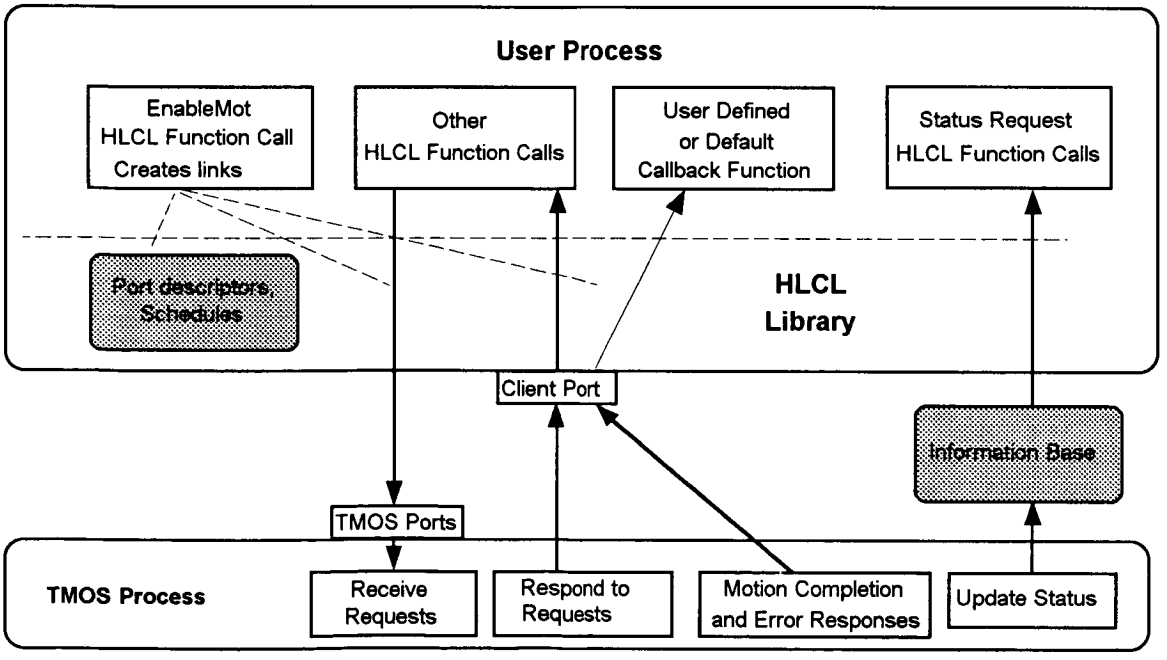


Fig. 1 -- Connecting a user process to TMOS

secret, but merely to shelter programs from changes in its structure with future enhancements.)

In addition to several other pieces of data, a TMOSPos keeps configuration information, so that when such a position becomes the argument of a machine motion, redundant solutions for the position can be reconciled. HLCL calls that return position information will also return configuration information. The user may keep this configuration information with the data or choose to ignore it.

Schedules and Other Modal Parameters

Sophisticated motion control algorithms employ many user or system definable parameters (as many as 100). It is inconvenient for the user to have to specify every parameter with each motion request. This problem is typically solved with modal parameters or system-wide parameters.

In a multiprogramming environment for multiple machines, we must further consider whether a parameter is specific to a thread of execution or whether it is specific to a particular machine. For example, it is appropriate for each separate program to keeps its own default value of speed to be used with every motion request. However, since the tool definition refers to a current physical attribute of a machine, that characteristic must be shared globally with all programs.

A TMOS schedule includes the modal parameters of speed, acceleration, motion type (linear, joint, etc.), and motion termination type.

Other modal parameters used in TMOS include a base frame of reference definition (called *Frame*) and a tool frame of reference definition (called *Tool*). These modal parameters are defined inside the TMOS process itself rather than in a schedule, because they are specific to a particular machine rather than a thread of execution. *Tool* and *Frame* are set with specific HLCL function calls.

Coordinate Systems and Coordinate Frames

All Cartesian motions produced by TMOS are defined for a specific Tool Center Point (defined by a homogeneous *Tool* transformation) and are defined with respect to a specific user definable frame of reference (defined by a homogeneous *Frame* transformation). *Tool* and *Frame* transformations are kept modally within TMOS for each machine under its control. That is, *Tool* and *Frame* need to be set only once, and those values will be used for each Cartesian motion of the machine until the modal values are changed. Changes in *Tool* and *Frame* take effect only at the beginning of the next motion using *Tool* and/or *Frame*.

Dynamic offsets to both transformations can occur at any time during a motion that uses *DeltaTool* and/or *DeltaFrame*. Changes to either of these transformations will take place immediately after the HLCL function calls that change them.

Interpolation and Termination of Motions

TMOS trajectories are broken into two categories, those that are terminated at a user specified destination (called *destination terminated*) and those that are unterminated (called *vector*). Vector motions are terminated by a succeeding motion or by an HLCL function call. Vector motions are useful for user defined sensor termination (also necessary for manual motion implementation.)

For Cartesian motions, the trajectories are of the Tool Center Point (TCP). In some systems, the destination position is checked for reachability of the tool center point before the motion is attempted. However, for many machines with nonlinear kinematics, this does not guarantee reachability of all positions on the trajectory. Therefore, TMOS does not check for reachability of destinations. TMOS confines itself to dynamic testing of reachability of each position on the desired trajectory for both *vector* and *destination terminated* motions. The test is made one deceleration period ahead of the machine's current position on the trajectory, so that the machine can be stopped on the trajectory just prior to the offending position.

The vector motions continue forever until stopped by a joint limit or are canceled or aborted. Many types of vector motion are available in TMOS, including move-to-joint-vector, move-to-world-vector, move-to-frame-vector, move-to-tool-vector, and move-to-wrist-joint-vector.

Destination-terminated motions are completed when the destination position of the motion is reached within the tolerance indicated by the termination condition. Motion types include joint-interpolated moves, straight line interpolated moves of the tool center point (TCP), circularly interpolated move of the TCP, straight line interpolation of the major axes in combination with joint interpolation of the wrist axes, and other more complex motion types.

All the above motions can be dynamically offset by various sensory inputs, such as vision.

Termination Conditions

The terminating condition of a motion for any of the above interpolation types determines how closely the machine must come to its destination before returning to the calling subroutine. TMOS provides a number of termination conditions, including:

- Return motion complete when the machine is within tolerance specified as *Fine* or as *Coarse* in the TMOS configuration.
- Return motion complete and start next motion when interpolation of this motion is complete (do not wait for servo tolerance).
- Start next motion when this motion begins deceleration. This provides the ability to blend motions.
- Initiate a new motion immediately when the next motion instruction is received. This is useful for vector motions in manual motion pendant implementation and user defined sensor applications.
- Other more complex termination types unique to TMOS, such as *fillet termination* which permits continuous motion at constant speed across motion segment boundaries.

Real-time Trajectory Modification

TMOS permits real-time modification of motions in progress. Routines are provided which accept a TMOS position as an incremental offset to either the part (*Frame*) or the tool position. This feature can be used to implement sensor-guided tracking for example. The C program can read the sensor in a loop and dynamically modify the motion in progress.

Arm Configurations

Since robots can generally reach a given Cartesian position in more than one way, Cartesian information alone is insufficient to completely determine the joint angles needed for a machine to reach that position. For example, the PUMA robot can generally reach a position with its wrist either above or below the elbow. Some redundant manipulators can reach nearly every position in an infinite number of ways.

The additional information needed to dictate how a machine will reach a given position is referred to as *configuration* information. Some controllers use specific commands to specify the configuration to be used in reaching a position. TMOS assumes configuration is part of the data. Therefore, all library routines that expect Cartesian input data also provide a parameter for configuration information.

For some simple machines, Cartesian positions can only be achieved in one way. Also, it is desirable to represent the positions of some objects without regard to how a machine might reach that object. Therefore, the *Euler* and *Transform* types defined for user representation of positions by the HLCL do not incorporate configuration information inside the data. Configuration data may be optionally added to these data types when they are converted to TMOS positions by the TMOS conversion functions, or when they are converted to machine joint angles.

Speed

TMOS by default provides coordinated motion of all axes of a machine on every motion. This means that for each motion request TMOS will coordinate the motion such that each joint of a machine will begin and end its motion at the same instant in time. The meaning of *speed* therefore, must apply not to individual axes, but to some entity which represents a combination of the axes of a machine. In some cases, speed applies to the tool center point with respect to the Frame. In some cases it is not possible to define a single point at which speed can be measured (joint interpolated motion for example) and speed is defined relative to some maximum. In addition, for motions which involve changes in both orientation and translation of the tool, both rotation and translation speed constraints must be taken into account.

TMOS considers up to three kinds of speed in the coordination of a motion, depending on the motion type:

- Tool translation speed - the speed of translation of the defined tool center point relative to the specified Frame.
- Tool rotation speed - the speed of rotation of one or more angles of orientation, measured in rotation units per second.
- Axis speed - the speed of motion of an axis (either rotation or translation), measured relative to the maximum for that axis.

Speed Limits

TMOS imposes speed limits only on a joint basis. That is, TMOS continuously monitors the speed of all axes. If any axis exceeds its speed limit as defined in the TMOS configuration file, then all axes are scaled back to maintain coordinated motion at the limiting joint speed. No limit is imposed on Cartesian translation or Cartesian orientation speed control.

TMOS also permits acceleration control on a per-motion basis.

Motion Commands and Continuous Motion

The HLCL is designed so that routines return as soon as possible. In the case of motion requests, the routines will return as soon as an "ID" can be assigned to the motion. The calling program can then continue processing other events and handle motion completion as an asynchronous event. If the programmer wishes to issue a series of motion requests, which is often the case, he must wait for previous motions to complete. A library routine is provided, which permits the caller to wait for the completion of a specific motion, identified by the ID.

Of course, the programmer can also cause continuous motion, by using a suitable termination type, so that the machine will not decelerate at the end of each motion and not wait for the completion of the previous motion. Continuous motion with the appropriate termination type will cause succeeding motions to be smoothly blended with previous ones. Smooth blending of motions at *constant speed of the TCP* is a unique option in TMOS.

Summary

TMOS, when combined with C-WORKS and other utilities, forms Nomad, a foundation for open architecture motion control for robots and other machines. Nomad software runs in a UNIX environment on off-the-shelf hardware, to provide low cost and high functionality motion control, standardized and yet amenable to tailoring for highly specialized applications.

AIAA-94-1254-CP

TELEROBOTICS FOR DEPOT MODERNIZATION

M. B. Leahy Jr **S. B. Petroski**
AFMC Robotics and Automation Center of Excellence
Technology and Industrial Support Directorate
Specialized Engineering Branch, Advanced Process Technology Section
San Antonio Air Logistics Center, Kelly AFB, TX 78241
ti-race@sadis05.kelly.af.mil

Abstract

The time is right to transition telerobotics beyond the traditional hazardous environment domain into industrial repair and remanufacturing applications. Air Force depots are prime examples of an industrial environment where small batch sizes, feature uncertainty, and varying workload, conspired to make classical industrial robotic solutions impractical and telerobotics a key enabling technology. The AFMC Robotics and Automation Center of Excellence (RACE) has launched the Unified Telerobotics Architecture Project (UTAP) to champion the development of the support infrastructure necessary to foster creative development and innovative utilization of emerging telerobotic technologies for depot applications. The objective of this paper is to demonstrate that telerobotics is a viable solution to a wide range of dual use applications, highlight the benefits from a unified approach, and provide an overview of the UTAP.

1 Introduction

The United States Air Force has five major Air Logistic Centers (ALC), or depots, that perform periodic weapon system maintenance. A significant portion of the periodic maintenance workload involves repair and remanufacturing. The small batch sizes, feature uncertainty, and varying workload that characterize the depot remanufacturing environment conspire to make classical industrial robotic solutions impractical for a wide range of depot processes. The robotics and artificial intelligence necessary to solve those problems with a completely automated system is beyond our grasp technically and economically. An equally demanding constraint is applied by a depot level workforce resistant to complete automation, and a management structure soured by the unfulfilled hyperbole of past robotics projects. But the requirement for robotic/automation based solutions is growing. New processes that are environmentally safe, but too demanding for human operators, the need for increased process consistency with lower manufacturing tolerances, and competition with industry all point to a larger role for judicious application of advanced robotics technology. The critical missing element is a method to bridge the gap, both culturally and technically, between manual operation and full automation. Telerobotics provides the means for building that bridge.

We broadly define telerobotics as the *technologies and systems that permit a human operator to direct and/or supervise the operation of a remote robotic effector mechanism* [1, 6]. Telerobotics does not imply a particular solution, but rather encompasses the whole range of application driven solutions ranging from telepresence to supervisory control. The key premise is to augment, not replace, the human operator by blending the individual abilities of each *system.* Humans have superior cognitive and pattern recognition skills, while the robot is a tireless precise positioning system. The telerobotic system is not a threat to job security, but rather a new innovative tool that adapts to the operator to maximize productivity.

[0]This paper is declared a work of the U.S. Government and is not subject to copyright protection in the United States

Unfortunately, telerobotics is more of a concept than an off-the-shelf technology. The basic components are available, and prototypes exist in various forms in numerous laboratories. However, developmental efforts have been targeted toward undersea, space and nuclear material handling applications. Solutions tend toward point designs customized to the particular application. Development of low cost systems was not a priority. Viewing the existing telerobotics market as a small niche, the major robot vendors have been reluctant to expend the resources necessary to modify their control systems to support a broad range of telerobotic solutions. Consequently, the manufacturing sector has been slow to embrace telerobotics and efforts to transition the technology from the laboratory to the shop floor are in their infancy. Therefore, we are presented with the unique and compelling opportunity to significantly influence the development of an emerging technology with the potential to radically enhance the productivity of the depot, and industrial, remanufacturing processes. The challenge is to implement the lessons learned from the mistakes of the past, to change our robotics technology insertion philosophy. Instead of developing *one-of* systems, we must embrace the creation of a unified systems concept that supports a large range of applications, provides an evolutionary path for incorporating new technologies, and reduces life cycle costs.

The Air Force Materiel Command Robotics and Automation Center of Excellence is championing the development of a unified framework or infrastructure that supports judicious insertion of telerobotics technology. The intent of this paper is threefold. First we present the case for telerobotics as a key enabling technology for depot process ranging from large aircraft paint stripping to surface finishing of component parts. Section three highlights the benefits of utilizing an unified architecture (infrastructure) to implement process solutions. In section four, our efforts to make that unified infrastructure a reality via the Unified Telerobotic Architecture Project (UTAP) are discussed. Conclusions are in section five.

2 Why telerobotics?

The best way to present the case for telerobotics for depot modernization is to overview the requirements for several target applications. Previous papers have presented detailed discussions of the telerobotic solutions to aircraft skin repair and fuel tank sealing/desealing [4, 5]. The remainder of this section is devoted to overviews of two processes targeted for prototype development under the UTAP. Specific process requirements (angle of incidence, standoff, accuracy) are in [6].

2.1 Aircraft Corrosion Control

At predefined intervals, aircraft are flown to the depots where existing paint is removed to allow surface inspection and repair of any corrosion damage. Before returning to active service, corrosion inhibitors are applied and the airframe is repainted. The productivity of all three processes; stripping, inspection, and painting can be improved by insertion of telerobotic systems. Paint removal is the initial target application in this area.

Process engineers responsible for corrosion control are being drawn to robotic systems due to efforts to eradicate the chemical stripping processes. Alternative paint removal techniques, while not environmentally hazardous, can be unsuitable for human application. High pressure (18K psi) water jet, CO2 ice pellets, flash lamps and lasers based application tools must be mounted as robot end-effectors. Even ignoring the obvious physical dangers, the application tools are too heavy (50 lbs or greater) for continuous human operation. Plastic Media Bead (PMB) and sodium bicarbonate blasting can be performed by operators in special air breathing suits, but the task is monotonous and messy. Another automation driver is the desire for stringent processes control. Many of the alternative stripping methods remove paint by blasting the aircraft or part with some media. Blasting introduces stress into the surface leading to reduced fatigue life. Tight control of the blasting process is necessary to minimize those side effects. Robotic systems provide a level of process control superior to that of a human operator. The unfriendly application environment, heavy payload, repetitive non-contact task nature of the task, and requirement for tight process control make the paint stripping operation ideally suited for robotic intervention.

The USAF has sponsored the development of large robotic paint stripping systems at three ALCs. Southwest Research Institute (SwRI) developed a custom system that is being used to strip F-16 aircraft with PMB and is being retrofitted for CO2 blasting of F-15s [7]. The Large Aircraft Robotic Paint Stripper (LARPS) REPTECH project is a large SCARA arm riding up and down on a column attached to an automated guide vehicle (AGV) [3]. The end-effector is a new commercial robot using a high pressure water process. Both SwRI and LARPS are big (50K lbs), expensive (>$2M), fully automated systems. But those processes are part of a large overall process that does not lend itself well to automation, ie the masking and general preparation of the aircraft for painting/stripping. At least half of the total process remains very manpower intensive. Add in the fact that several installations already have stacker (telecrane) platforms that allow human operators to access large portions of the aircraft surface and one can make a compelling argument for augmenting the existing workforce instead of replacing it.

A telerobotic aircraft paint removal scenario would look like the following. Attach a small robotic end-effector to the underside of the telecrane. The operator manually drives the stacker crane into the proper stripping position and then uses a joystick and the robots force sensing capability to register the actual worksite to a predetermined stripping trajectory. After setting stripping and other application parameters the operator becomes a supervisor as the system autonomously executes the stripping process. To perform the process the system must maintain a stripping process dependent separation/standoff distance and a tooling angle of incidence to the workpiece normal. While the primary mode of operation is supervisory, the system shall support a shared control feature that slaves end-effector position to the joystick with the system automatically regulating standoff and angle of incidence so that the operator can quickly remove any excess paint left by the autonomous process.

The robotic system is not responsible for all paint removal. The human operators, necessary for the preparation, would still be utilized to strip hard to reach locations. But the new tools free the operator from directly applying the stripping process to over 80% of the aircraft while dramatically improving process control. A properly designed telerobot system will support all stripping processes. Switching to painting and inspection tasks only requires some quick change tooling. Attached to mobile lift platforms the same system could perform flight line touch-up, or cross over to dual use applications like highway bridge repair.

2.2 Surface finishing

The standard procedure for repairing dents in engine nacelles is to fill the indentation with a fiberglass epoxy compound and then finish the surface to the required smoothness. Repair of aluminum-honeycomb aircraft skins frequently requires a similar blending process around the seams of the patched section. Grinding is also employed to remove the paint in the vicinity of the repair site. The common theme in these, and many other backshop operators, is the utilization of manual sanders and grinders. The health risks imposed by repetitive motions and dust inhalation combined with requirements for stricter process control and repair of more exotic composite parts are driving the search for incorporation of robotic technologies.

The customer does not consider the old approach of tight fixturing and preprogrammed motions an option. Management does not want to replace workers, but rather make them more productive and provide a safer environment. What is mandated is a better tool to replace the current hand sanders and polishers. Telerobotics provides that tool.

To augment the surface finishing task, a telerobotic system must support the following functionality. Instead of holding the hand tool, the operator grasps an input device (possibly a force reflecting joystick) that commands a robot permanently attached to the shop floor. Work pieces are still clamped onto dollies and rolled into the robot's work area, but no additional fixturing is required. Through a quick change mechanism the system is capable of matching the tooling to the task. The operator drives the robot into contact with the surface and performs a series of motions to complete the task under two shared control modes. In mode one the system maintains a

contact force and a tooling angle of incidence to the workpiece normal. In mode two the system maintains a tooling angle of incidence to the workpiece normal while allowing the operator to modulate the applied contact force. Commands that would result in a contact force exceeding a predefined limit are automatically regulated at the limit. Both modes must be supported without any a priori knowledge of part geometry. However, the system must be flexible enough to efficiently incorporate automatic trajectory generation software when it becomes commercially available. Dual use applications of this technology range from polishing of bathroom fixtures to removing machining marks on airframe skins and ship impulsers.

3 Why a unified approach?

For a judicious insertion to take place one must specify the proper level of technology and deliver a system specification that is cost effective. The true potential of telerobotics can not be realized if every application requires a costly custom solution. A unified infrastructure for telerobotics is driven by the overriding objective of reducing system life cycle costs. Insertion cost decrease as supportability and reliability increase along with ease of upgrading.

3.1 Insertion Costs

Under the custom solution approach, software development and system integration are at least 60% of a new insertion project and almost always the bottleneck. A common framework allows basic commands for movement, gripping, trajectory generation, obstacle avoidance, and operator interface, etc to be developed at a higher level of abstraction. After paying for the initial software development, the scope of the software development task is reduced to developing the specific code that is required to implement a new process. Phase two of the UTAP will validate our estimate that initial development costs can be amortized within the first three applications. JPL estimated that a unified architecture could be reconfigured for a new application in one manweek.

3.2 Upgradability

The government procurement process requires that we rigorously specify the functional requirements of any system we contract for. The standards and specifications we mandate must be achievable by multiple vendors to allow full and open competition. Without standards we can not remain competitive as technology advances. Standardizing at the interface level, provides the hooks and scars for future upgrades without limiting the contractor's freedom to provide the most innovative and cost effective solution. For example, replacing a trajectory generator module must not require an extensive software rewrite because the existing generator is imbedded into some piece of spaghetti code. Switching joysticks should be no more complicated then switching printers on a computer system. By mandating standardization at the interface level we take the first step toward full interoperability. A unified architecture supports a system design methodology that evolves as the culture and technology evolve by providing a framework that builds in the future instead of locking it out.

3.3 Supportability

A common infrastructure breaks the one robot, one technician, one programmer, single operator loop we are currently trapped in. A unified architecture permits a common operator interface, reducing training requirements. The higher level of abstraction eliminates the need for programmers to be fluent in multiple robot languages, again reducing training time and expense. Adding a new system into an existing facility no longer mandates the creation of a whole separate support hierarchy. Upper level support is easily centralized. By avoiding custom mechanism designs, hardware maintenance support costs are also dramatically reduced and are now available from a variety of sources. A single internal organization will provide technical support for a whole depot. The need for an expensive support contract, usually with the original manufacturer of the custom system, is eliminated.

As the size of our workforce continues to decrease, increasing the productivity and range of skills of individual operators becomes more important. A single operator must become proficient in numerous processes and the robotic systems that are embedded in them. A common infrastructure will support a com-

mon operator interface allowing a seamless transition across all the telerobotic systems in the depot. Upon login the system will autoconfigure the look and feel to match known operator preferences.

3.4 Reliability

Software is the most unreliable portion of robotic systems. A common architecture allows a majority of the software to be ported from one application to the next. Minimizing the creation of new code maximizes system reliability. Selection of proven hardware components mitigates mechanical breakdown.

4 UTAP Overview

The Robotics and Automation Center of Excellence (RACE) has embarked on a multi-year initiative to demonstrate the feasibility of telerobotic technologies to accomplish a wide range of manufacturing applications and to develop a unified architecture that radically reduces the life cycle costs of telerobotic systems. The Unified Telerobotic Architecture Project (UTAP) is tightly coupled to related efforts in the national labs and the domestic manufacturing industry to maximize leveraging and dual use technology transfer opportunities.

In Phase 0, completed in FY93, NASA's Jet Propulsion Laboratory (JPL) performed an engineering study to define a telerobotic architecture capable of performing a wide range of ALC remanufacturing applications. The study began by distilling a representative set of processes into a global set of functional requirements sufficient to span the needs of depot activities. The state of commercial and near-commercial technology was then surveyed to determine how these requirements may be met in an integrated system. A comparison of the functional requirements and the available technology products then produced an architecture of system components and their connectivity [1, 6]

RACE has tasked the National Institute of Science and Technology (NIST) to act as coordinator and prime contractor for the FY94 study of issues pertaining to the specification and validation of the architecture.

Phase 1, currently underway, is a joint effort by NIST and JPL to examine the feasibility of implementing the initial JPL architecture and to develop preliminary interface specifications between all functional blocks of the UTA. This effort will include consideration of telerobotic technologies being developed at national labs and emerging standards such as the Next Generation Controller Specification for an Open System Architectural Standard. A workshop will be held with industry and national lab representation to solicit input for the validation and consolidation of these preliminary interfaces into the UTA design. The output of this workshop will be a working document that describes the interfaces and functional blocks of the UTA for Phase 2.

In Phase 2, a systems integrator under contract to NIST shall be tasked to analyze the UTA interface specification and determine if an UTA compliant system can be implemented to solve the representative application set, or suggest modifications to the portions where compliance is not possible. The contractor will then validate their analysis by designing an UTA compliant system and performing the validation test set, which consists of:

- Autonomous regulation of separation/stand-off while the human operator controls the other two cartesian coordinates via joystick,
- Autonomous force regulation along a gently curved surface while the other two tangential cartesian coordinates are are controlled via joystick,
- Registration of a workpiece by use of a vision system and fiducials,
- Autonomous regulation of tooling angle of incidence to the workpiece normal, and,
- Vision based tracking of circular trajectory on a planar surface.

The contractor will demonstrate accomplishment of the tasks on physical hardware using an Adept motion servo system and then demonstrate the interoperability and modularity of the architecture by replacing the Adept system with a Trellis motion servo system. Specific designs for three prototype systems

and an estimate of system integration costs and potential cost savings from a unified approach are also required.

Future phases of the UTAP are not as crisply defined, but the objective is to continue UTA refinement to the goal of a releasing the architecture specification in full system request for proposals in FY97. Phase 3, the prototype development phase, will see contracts awarded to systems integrators to implement the Architecture/Interface specifications as depot prototypes. Each selected process will demonstrate a different facet of telerobotic technologies and will provide a core capability of the system. The three projected applications are: Telerobotic Telecrane Paint Stripping (T2PS), Telerobotic Surface Finishing (TSF), and the Telerobotic Cutting System (TCS). A parallel effort to create more sophisticated *laboratory* prototypes which exercise even more of the potential of telerobotics is also anticipated. Currently our prototype center PUMA is being retrofitted with more commercial version of the Onika software environment developed at Carnegie Mellon University [2]. Fitted with a force and vision system, the enhanced PUMA will be used to investigate the advantages of full interoperability in a telerobotics environment. Phase 4 encompasses a 6 month operator prototype evaluation and analysis task. Throughout the prototyping and operator evaluation phases lessons learned will be feed back to produce a more robust architecture specification.

5 Conclusion

The problem is enhancing the quality of Air Logistic Center repair and remanufacturing processes. The constraints are technical, economical, and cultural. Creative development and innovative application of telerobotics technology is the solution. The challenge is to redirect our system design philosophy to a methodology that embraces integration of commercially available components under a unified telerobotic architecture or framework. In cooperation with other national laboratories and agencies the AFMC Robotics and Automation Center of Excellence is championing the development and prototyping of a unified architecture that will pave the way for judicious insertion of telerobotic systems into a wide range of dual-use applications.

References

[1] P. G. Backes, W. Zimmerman, and M. B. Leahy Jr., Telerobotics Application to Aircraft Maintenance and Remanufacturing, submitted to the 1993 IEEE ICRA.

[2] M. W. Gertz, D. B. Stewart, and P. K. Khosla, A Software Architecture-Based Human-Machine Interface for Reconfigurable Sensor-Based Control Systems, *Proc of the IEEE Int Symp on Intel Control*, Chicago, IL, Aug 25-27, 1993, pp. 75-80.

[3] S. Hofacker, Large Aircraft Robotic Paint Stripping, *Proceedings of the SME Conference on Maintaining and Supporting an Aircraft Fleet*, Dallas, TX, Jun 9-11, 1992.

[4] M. B. Leahy, Jr. and B. K. Cassiday, RACE Pulls for Shared Control, *Proceedings of SPIE Conference on Cooperative Intelligent Robotics in Space III*, Boston, MA, Nov 16-18, 1992.

[5] M. B. Leahy, Jr., and S. B. Petroski, Telerobotics for Depot Applications, *Proceedings of NAECON 93*, Dayton, OH., May 1993.

[6] *Generic Telerobotic Architecture for C-5 Industrial Processes*, JPL Final Report, Aug 93.

[7] *Robotic Paint Stripping System*, Southwest Research Institute Yearly Report, 1992.

AIAA-94-1255-CP

A SMART TELEROBOTIC SYSTEM DRIVEN BY MONOCULAR VISION

R.J.P. deFigueiredo † A. Maccato † P. Wlczek † B. Denney ‡ J. Scheerer ‡

†Dept. of Electrical and Computer Engineering, University of California, Irvine CA 92717
‡McDonnell Douglas Aerospace, 5301 Bolsa Ave, Huntington Beach, CA 92647

Abstract

There is an increasing demand for space robotic systems which can reduce the number of potentially hazardous EVA's on manned space missions. In addition, telerobotic maneuvers can easily become long and tiresome for the operator. This paper describes a robotic system which accepts motion and control commands which can be generated autonomously.

The system developed has been designed to perform an autonomous grapple based on guidance control feedback provided by images from a single camera mounted on the slave robot's end effector. The vision system consists of three parts. The first part is signature based, trained on an arbitrary grapple interface (i.e. no special targets are required for guidance); it provides estimates for the 3D attitude of the interface by interpolating sampled signature correlations. These signatures are essentially the distribution of line orientations obtained by radial integration of the Fourier transform of a pre-processed edge image. The second part estimates the range and bearing of the interface based on the first and second moments of the preprocessed edge image of the interface. And the third stage of the algorithm verifies the results.

The robot path follows a linear translation trajectory which is repeatedly adjusted for errors via the vision system. The end effector's attitude is adjusted along the trajectory such that the grapple interface always remains in center view of the camera.

Introduction

Teleoperations are becoming increasingly important in hazardous environments (e.g. chemical plants, nuclear power plants, space). Space systems applications, such as space-based assembly and maintenance, automatic rendezvous and docking, space exploration, and satellite monitoring and tracking [1] are of particular interest due to potentially long delay times between operator and robot. For instance, it has been estimated that robotic operations can take several times as long as extra-vehicular activity (EVA) to perform similar tasks [2,3]. Long delay times and limited bandwidth require the robot to accept only high level commands and to possess locally a certain degree of autonomy.

Object recognition and attitude determination of objects are essential components for successful sensor based teleoperational semi-autonomous robotic systems. This paper will cover camera based systems, due to the relatively low cost of CCD cameras and their wide use in remote robotic systems.

Current vision based robotic systems utilize visual guidance targets. These targets must be placed on objects with which the robot is to interact[4]. However, when the objects are not readily accessible to humans, which is the case when operating in a hostile environment such as space, the system restricts the class of robotic interactions to those which are specifically identified and designed a priori.

The new vision system developed eliminates the need for these guidance targets by allowing the object, or part of the object (i.e. a grapple fixture), to become the robot's visual guidance target. This is accomplished by teaching the vision system the object by presenting different views. This training could be done with a physical object or by using a CAD model of the object.

The complete description of a particular target relative to the camera consists of six parameters: roll, pitch, yaw, range, and two bearing parameters. All six can be estimated, in principle, from a single camera image and knowledge of the target's solid geometry.

We have developed a new technique for determining the three-dimensional roll, pitch, and yaw attitude target parameters and the three translation parameters assuming that the object is known and unoccluded.

Method

We restrict the class of images to those of machined objects, which characteristically produce sharp edge discontinuities. The edge discontinuities result from the projection onto the image plane of the polytopes, cylinders and conic sections comprising the object. Our approach relies on these projected edges as the basic features required to analyze and interpret the image data.

Attitude Estimation

The technique for estimating the attitude relies on extracting a signature of the object as viewed by the camera, and then matching it against signatures of the same object with known attitudes, generated off line

Copyright ©1993 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

from a model of that object. The attitude estimate is obtained by interpolating among the signatures with the highest matching scores. The overall procedure is diagramed in Figures 1.

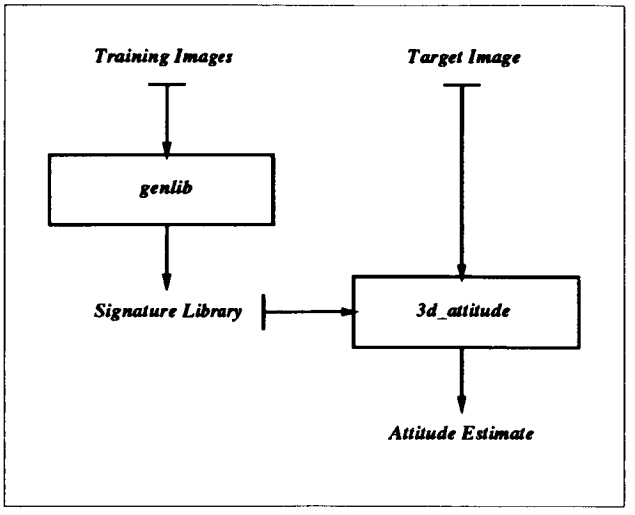


Figure 1: Overall data flow diagram for the estimation of the attitude parameters.

The algorithm computes as a signature the distribution of edge segments in the image as a function of orientation in the image plane. When an object undergoes an attitude transformation, the distribution of line orientations in the image plane changes; therefore, the signature contains implicit information about the object's attitude. On the other hand, the signature is insensitive to the range and bearing of the object, as these do not affect the distribution of line orientations in the image. The signature matching procedure approximates the inverse map from line orientations to object attitude.

The signature extraction computation involves three steps. First, a binary line image is obtained from the original picture (Fig. 2), reducing the effect of changes in illumination of the target (Fig. 3). The preprocessing requests identification of the object within the field of view, and removal of clutter in the image. We achieve this by an image segmentation strategy discussed in detail in the Appendix. The line image is then mapped into the two-dimensional Fourier domain, effectively collapsing range and bearing information, while preserving information on the object's roll, pitch, and yaw (Fig. 4). Lastly, a weighted sum of the magnitude in the Fourier image yields the distribution of line segments as a function of orientation, which serves as an attitude signature (see Gonzales and Wintz[5] for an introductory discussion on the properties of the Fourier Transform applied to image processing)(Fig. 5).

In particular, the Fourier transform provides an efficient and robust means of extracting the signature. In essence, any straight line in the image plane is mapped by the Fourier transformation into a straight line passing through the origin of the transform domain, and orthogonal to the original line. The distance from the origin of the original line results in a complex phase modulation of its transform. By linearity of the Fourier mapping, an image consisting of several straight lines is transformed into a superposition of lines emanating from the origin. Thus, a radial integration of the Fourier transform's magnitude, about the origin of the transform domain, yields the desired signature. To compensate for the finite thickness and length of actual line segments in the image, the Fourier transform is radially weighted, to deemphasize edge thickness.

The attitude parameters are found by performing a cyclic cross-correlation of the target signature with the library signatures and selecting the maximally correlated match. The best signature picked reflects the object pitch and yaw. The offset of that signature match reflects the roll. Since signatures are 180° symmetrical there is a 180° ambiguity in the roll measurement. This ambiguity will be resolved in the match verification process described in Section 2.3.

Position Estimation

The technique for estimating the range and the two bearing parameters of the object relies on the center of gravity x_c, y_c, and the sum of the variances σ_o^2 along the x-axis σ_x^2 and the y-axis σ_y^2 of the object's edge image:

$$\sigma_o^2 = \sigma_x^2 + \sigma_y^2 \tag{1}$$

The range of the object is determined using σ_o^2. It can be shown, that σ_o^2 is invariant to rotation and translation of the image [6-8]. Using a perspective projection, and assuming that the size of the object is small compared to the range, the distance of the object in the actual image, z_0, is given by,

$$z_0 = \frac{f_0 \times \sigma_{\text{ref}} \times z_{\text{ref}}}{f_{\text{ref}} \times \sigma_o} \tag{2}$$

where σ_o is the square root of the variance of the actual image, f_0 is the focal length of the lens used, σ_{ref} is the square root of the variance in the edge image of the matching signature, f_{ref} is the focal length of the lens used in generating the signature library, and z_{ref} is the range of the object used during training.

By knowing the deviation of the center of gravity of the actual edge image against the center of gravity of the edge image of the matched library signature, the two bearing components are determined by:

$$\rho = \tan^{-1}\frac{x_1}{f_0} - \tan^{-1}\frac{x_{\text{ref}}}{f_{\text{ref}}} \tag{3}$$

$$\phi = \tan^{-1}\frac{y_1}{f_0} - \tan^{-1}\frac{y_{\text{ref}}}{f_{\text{ref}}} \tag{4}$$

where (x_1, y_1) and (x_{ref}, y_{ref}) are the center of gravities of the actual edge image and the training edge image respectively. ρ and ϕ are the values of the bearing parameters along the y-axis and x-axis respectively.

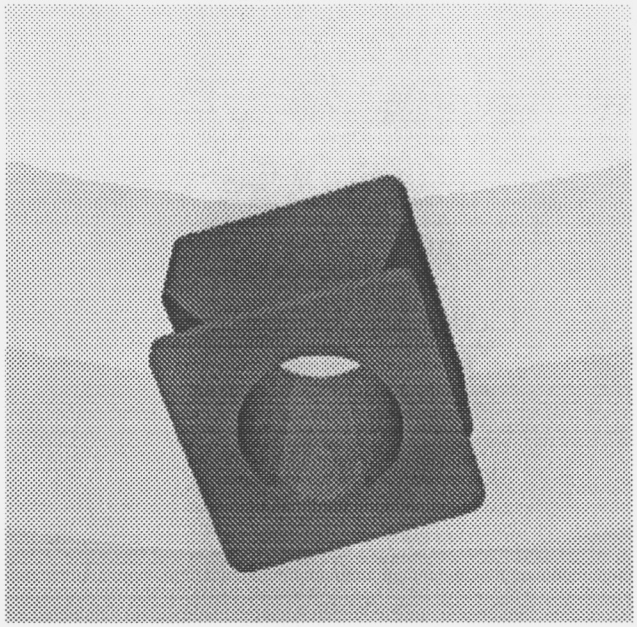


Figure 2: Synthetic image of the Micro Interface device, a typical machined object.

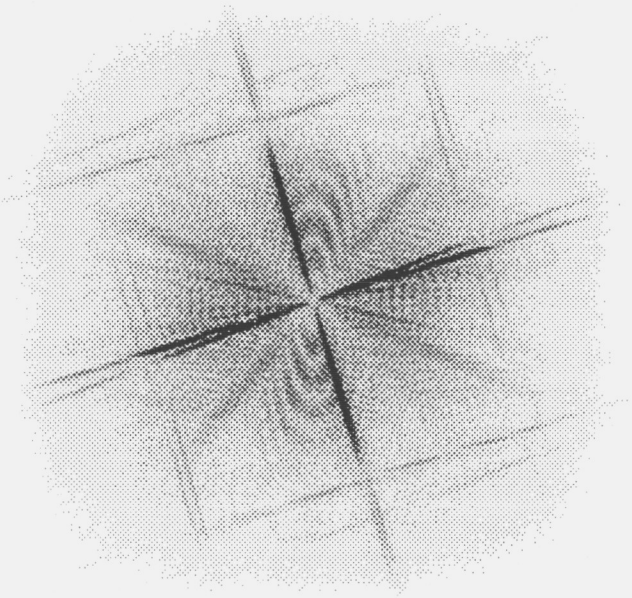


Figure 4: The weighted 2D FFT transform of the edge image of the Micro Interface Device.

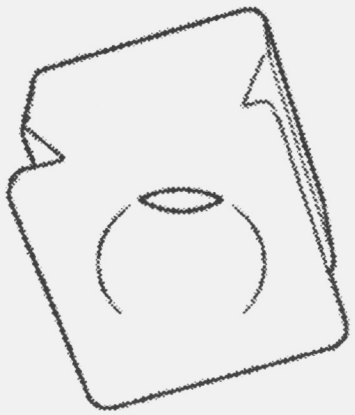


Figure 3: The edge image for the Micro Interface device.

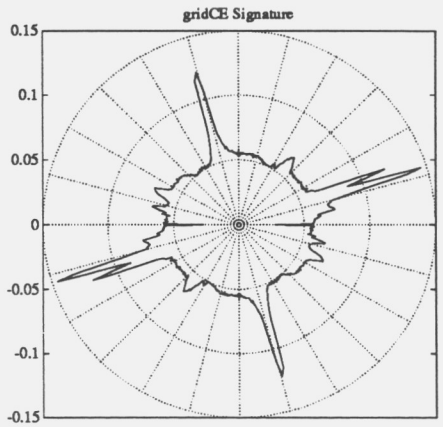


Figure 5: The extracted signature, encoding the distribution of line edge orientations in the original image of the Micro Interface device.

Model Based Attitude Estimation and Verification

The six attitude parameters found in Sections 2.1 and 2.2 must be verified and the ambiguity of the roll must be resolved. This is accomplished with the help of a perspective projection (overlay) of a three-dimensional model of the target (Figure 6) in a cross correlation with the edge image of the object seen by the camera. The overlay with the highest correlation yields the best estimate of the attitude and position of the target.

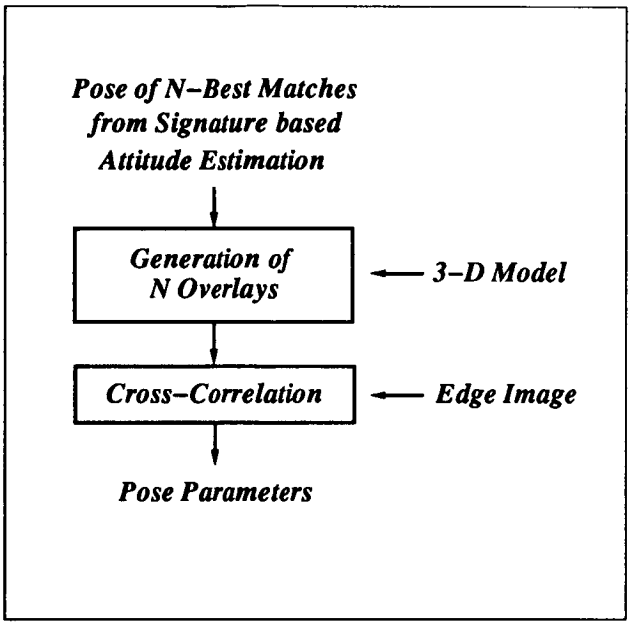


Figure 6: Data flow diagram for estimating the pose of an object based on the projection of a three-dimensional model of the target

The six pose parameters of the n-best matches from the signature based algorithm in Section 2.1 were used to generate n corresponding overlays. A typical overlay is shown in Fig. 7. Those overlays were matched

The three-dimensional model of the object was defined in terms of polygons where each polygon was derived by its vertices. To each polygon a surface normal was assigned to calculate the visibility of the polygon for the current attitude of the object. The visibility check was achieved by determining the sign of the dot product between the normal vector and a vector extending from the the polygon to the view point. For positive values the polygon was visible and for negative values invisible.

To increase the robustness and precision of the cross correlation we correlate the directions of the edges with the direction of the overlay edges. By looking at the directional image gradient we obtain not only the strength of the edge but also its direction. The

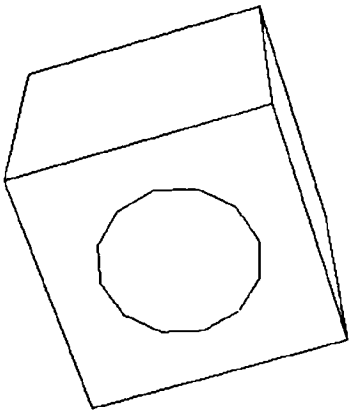


Figure 7: Example of an overlay which was used in a cross correlation to estimate the pose of a target (The jagged edges are caused by the typ-setting process).

modified cross correlation can be stated as

$$match(i,j) = \sum_{x=0}^{N_x} \sum_{y=0}^{N_y} \left| \langle \vec{pt}_{im}\ (x,y)\ \vec{pt}_{ov}\ (i+x, j+y) \rangle \right| \tag{5}$$

where $\langle \cdot \rangle$ denotes the dot product between two vectors. The vectors $\vec{pt}_{im}\ (x,y)$ and $\vec{pt}_{ov}\ (x,y)$ are defined as the two-dimensional vector $[\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}]^T$ from the camera image and overlay image respectively. It has to be noted that in Equation 5 we only have to perform the cross correlation in the vicinity of the projection of the object model because the signature based algorithm gives reliable estimates of the position of the target.

The combination of the signature based method shown in Section 2.1 and the above approach based on cross correlation allows us to overcome one of the main disadvantages of the model based methods shown in the literature [9-10] where a correspondence had to be established between image features and model features to solve for the attitude parameters. With the signature based algorithm we are able to prune down the search tree of possible aspects of the model and reduce the range of the cross correlation considerably.

Robot Control

The algorithm for moving the robot towards the target to perform a grapple is described below:

- Using a camera mounted on the end effector, estimate the position and attitude of the target (i.e the handle to be grappled) with respect to the

camera. We will call this frame $\hat{H}$ where the term frame refers to both attitude and position.

- We can define a frame, C_{FH}, with respect to the target, which is the desired final frame of the camera for the approach. Now we can use our estimate for the target to come up with an estimate for C_F, called $\hat{C_F}$, with respect to the Camera frame C. If C is within tolerable limits of C_F then we are at the final position and attitude and can grapple the object.
- If we have not reached C_F then we can calculate our next desired camera position by using the following constraints on the next camera frame, denoted by $N(C)$.
  1. The origin of $N(C)$, denoted by $N(C)_0 = N(C_0)$, falls on the line $\overline{C_0 C_{F0}}$.
  2. $N(C)$ should be at most a distance of $d_{\max}$ from C.
  3. The z-axis of $N(C)$, denoted by $N(C_z)$ should point towards $\hat{H}_0$.
  4. $N(C_x)$ should be perpendicular to both $N(C_z)$ (of course) and $\hat{H}_y$. In particular the sign of the vector is defined by $N(C_x) = \hat{H}_y \times N(C_z)$.
- We can calculate $N(G)$ with respect to G from $N(C)$, since the relationship of the camera frame C to the end effector frame G is known. This information can be put in the form of relative (x, y, z, R, P, Y) moves.
- Command the robot to make the relative move calculated above.
- Repeat the entire process.

Results

We have tested the algorithm on a set of synthetic images of an interface device used in space system applications (Fig. 2). The Micro Interface device is used in SSF robotic operations. A ray-tracer was used to generate the synthetic images' aspect transformations of the target with respect to the image plane. Although the results presented in sections 3.1-3.4 were generated with synthetic images, similar results have been obtained for real camera images.

A 5×5 signature library was generated from synthetic images to cover a square patch 10° on the side in the pitch–yaw plane with an inter-signature separation of 2.5° in each direction. The center orientation was selected to correspond to a typical view of the Micro Interface during a grasping operation. This signature library was representative of more realistic libraries covering a larger range of pitch and yaw parameter values.

Using this signature library, four tests were performed:

1. Random roll, pitch and yaw attitude estimation.
2. Bearing and Range estimation.
3. Range invariance test of roll, pitch and yaw.
4. Bearing invariance test of roll, pitch and yaw.

For consistency with the ray-tracer program, the target's attitude in all four tests was represented using three Euler angles, which measure attitude through a set of three rotations about the z, x, and again z axes, in the camera's frame of reference (the image plane coincides with the xy-plane, and faces the negative z axis). We denote these three rotation angles by α, β, and γ, respectively. The translation components, range and bearing, of the image plane around the x-axis and y-axis were denoted with z, ϕ, and ρ respectively.

The tests provide evidence for the viability of the approach to 3D attitude and position determination. The procedure accurately estimates the position and the three attitude parameters of the object. The algorithm shows invariance to the range and bearing of the target for the estimation of the 3D attitude. These test results are described in the sections 3.1-3.4.

Roll, Pitch, and Yaw Estimation

A random set of 10 target images with three arbitrary Euler angles was used to test the algorithm's ability to correctly determine the target's attitude. The exact and estimated Euler angles are shown in Table 1.

The average error in any one parameter is 0.6°. The maximum error occurred for the α parameter of Image J, a difference of 2.7°. For this image, the wrong library signature was selected in the matching stage. The difference in the γ parameter partially compensates for this error, reducing the combined $\alpha + \gamma$ angular error for this image to only 1.2°.

Bearing and Range Estimation

A set of 4 target images was used to test the accuracy of the procedure for the bearing parameter and a set of 6 target images was used to test the accuracy for the range. The exact and estimated parameters for range and bearing are shown in Table 2 and Table 3. The average error is 2.2cm for the range estimate and 0.1° for the estimate of the bearing.

Range Invariance

A set of 14 target images was used to test the algorithms sensitivity to the target's variation in range. The range of the target in the training images, used to generate the signature library, was 30cm from the image plane. The exact and estimated Euler angles are

Table 1: Attitude estimation test results.

Image	Exact Angles (degrees)			Estimated Angles (degrees)		
	α	β	γ	α	β	γ
A	16.09	22.63	-8.55	16.57	22.15	-9.98
B	15.24	20.46	2.63	15.60	20.33	2.81
C	18.39	23.27	7.69	19.29	22.44	6.19
D	18.40	22.08	-4.55	17.91	21.89	-3.65
E	19.67	23.51	-1.27	20.57	23.11	-1.55
F	16.92	24.55	5.33	16.90	24.36	4.78
G	17.60	23.81	-0.45	17.86	23.51	-0.14
H	19.15	21.31	-5.24	17.95	21.38	-3.51
I	15.17	20.24	-4.50	15.32	20.01	-4.22
J	15.27	23.68	-2.81	12.50	24.74	-0.70

Table 2: Range estimation test results.

Image	Range (cm)	Estimated Range (cm)
A	31	31.17
B	35	35.70
C	39	40.00
D	45	46.30
E	60	62.10
F	90	98.00

Table 3: Bearing estimation test results.

Image	Bearing (degrees)		Estimated Bearing (degrees)	
A	1	0	1.00	0.02
B	2	0	2.00	0.05
C	3	0	3.07	0.13
D	4	0	4.14	0.18

shown in Table 4, for various target ranges. Angles are measured in degrees, range in centimeters.

The errors incurred are moderate, and degrade as the range increases. The maximum error occurred for the α parameter of Image L, a difference of 5.0°. For this image, the wrong library signature was selected in the matching stage. The difference in the γ parameter partially compensates for this error, reducing the combined $\alpha + \gamma$ angular error for this image to only 1.5°.

Table 4: Range invariance test results.

Image	Range (cm)	Exact Euler Angles (degrees)		
		α	β	γ
*	30	17.500	22.500	0.000
Image	Range (cm)	Estimated Euler Angles (degrees)		
		α	β	γ
A	30	17.553	22.342	0.000
B	31	17.455	22.199	0.000
C	32	17.514	22.193	0.000
D	33	17.463	22.116	0.000
E	34	17.467	21.989	0.000
F	35	17.436	21.827	0.000
G	36	17.468	21.533	0.000
H	37	17.492	21.529	0.141
I	38	17.412	21.257	0.141
J	39	17.521	21.351	0.000
K	45	18.068	20.050	0.140
L	60	22.500	17.500	-3.518
M	90	17.989	20.885	0.140
N	150	17.776	21.297	0.140

Bearing Invariance

A set of 5 target images was used to test the algorithm's sensitivity to the target's variation in bearing. The bearing of the target in the training images used to generate the signature library was 0° from the image plane's normal. The exact and estimated Euler angles are shown in Table 5, for various target bearings, away from the image plane's normal, in the direction of the positive y-axis. Both Euler angles and bearings are measured in degrees.

The errors incurred are moderate, with a maximum error in the β parameter of Image E, a difference of only 0.4°. Bearings of more than 4° would have brought the target partially outside the field of view of the camera, and were not tested.

Table 5: Bearing invariance test results.

Image	Bearing (degrees)	Exact Euler Angles (degrees) α	β	γ
*	0.0	17.500	22.500	0.000
Image	Bearing (degrees)	Estimated Euler Angles (degrees) α	β	γ
A	0.0	17.553	22.342	0.000
B	1.0	17.435	22.510	0.000
C	2.0	17.472	22.549	0.000
D	3.0	17.385	22.735	0.000
E	4.0	17.337	22.939	0.000

Verification Method Results

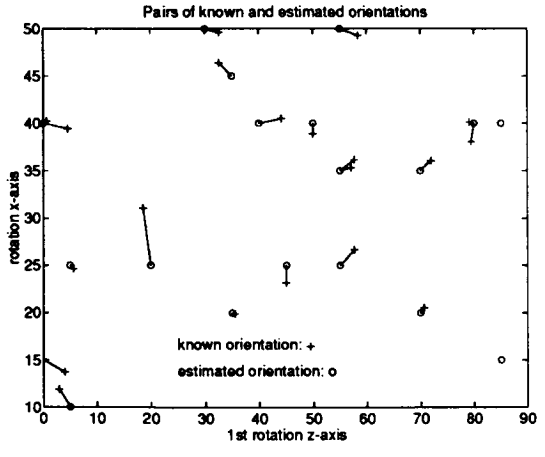


Figure 8: Matching random test points to best overlay candidate (candidates are on a 5 degree spaced grid)

Figure 8 shows the overlay matching results. Each random test point is marked with "+" and has a corresponding (correspondence is indicated by a connecting line) estimate denoted by "o". The estimates in this example fall on a $5° \times 5°$ grid. As seen by the figure, all but one of the matches fell on the nearest grid point (i.e. the estimates were within 5 degrees).

Conclusion

A procedure has been developed to determine the 3D attitude and the position of machined objects without the use of any special marks. Since there is no need for marks, an existing implementation of the algorithm can be quickly adapted to a different object, by supplying a signature library for the new target. Moreover it is not necessary to possess a physical model of the object because it is possible to generate the signature library with a ray-tracer program . In addition the signatures require only 1K bytes of memory each. Thus for a typical signature library of 225 signatures, the signature library is smaller than one 512 by 512 image.

The algorithm relies on standard image processing routines (e.g. edge extraction, 2D Fourier Transformation), which are available in numerous image processing libraries, and fast hardware implementations.

The 2D Fourier Transformation, which is the most time consuming part of the procedure is readily parallelized, so that a real time version of the algorithm can be achieved by distributed hardware.

Although this method was developed under the assumption that there is no clutter in the image and that the target is of a known type, these constraints can be lifted using additional initial scene analysis. For example, the scene can be segmented using standard image processing algorithms, and potential objects can be compared to known objects via signature matching and match verification.

Acknowledgements

This work has been supported in part by McDonnell Douglas Aerospace and the University of California MICRO program.

Appendix

In order to remove background clutter we use information about the constraints of our scene with respect to our target. This appendix discusses in detail the image preprocessing techniques which we used in connection with the robotic grappling application discussed in this paper.

Preprocessing the Raw Image

We start our processing given a single frame 256 gray scale image, I. The image I is filtered three times producing three more useful images. The first is a low pass filtered version of the raw image, denoted by $\tilde{I}$. The next two filtered images are the x and y gradients of the raw image, denoted as $\nabla_x I$ and $\nabla_y I$ respectively. Two binary edge images are then constructed using the above filtered images.

The first edge image is a thin edge image, E, found by,

$$E = \left\{ \left[\frac{\sqrt{(\nabla_x I)^2 + (\nabla_y I)^2}}{(\tilde{I} + 1/2)} \right] > \frac{1}{2} \right\} \qquad (6)$$

where ">" is treated as pixel-wise binary output operator. The above equation attempts to enhance local edge information by dividing the magnitude of the gradient of the raw image by the average neighborhood pixel intensity. Thus small intensity variations in dark regions could be equivalent to larger variations in lighter regions[11].

The second edge image is a thick edge image, E^+, defined as,

$$E^+ = \left\{ \left[\frac{\sqrt{(\nabla_x I)^2 + (\nabla_y I)^2}}{(\tilde{I} + 1/2)} \right] > \frac{1}{4} \right\}. \qquad (7)$$

The image E^+ is nearly identical to E except that the threshold used is lower. Thus, E^+ contains more *white* pixels (pixels which satisfy the binary condition). Therefore, $E \subset E^+$ (i.e. every white pixel of E is a white pixel in E^+). Note that while E provides a cleaner edge image, E^+ preserves the connectivity of the edge image. This connectivity will be used below to determine a processing region which rejects background clutter.

Rejecting Background Clutter

In the discussed application we are interested in finding a handle which is mounted on a predominantly lighter background. In addition we assume that the handle structure will be larger than any unwanted clutter on the same background. Thus, we look for the largest edge structure in a dark region which is contained in a lighter region. This region tells us which information in E should be processed and which information should be rejected. Next we must determine what is light and what is dark as well as what is considered an edge structure.

Using the original raw image, I, we generate a histogram. This gray level histogram is then clustered into three fuzzy classes, *dark pixels*, *medium pixels*, and *light pixels* by using a fuzzy c-means clustering algorithm[12].

The light regions of the raw image are found using the mid-point between the *dark* and *medium pixels* cluster centroids as a image threshold. This thresholded image is then segmented into blobs based on the pixels *8-connectivity*[5,13]. The connectivity analyses only reports significant blobs (blobs which contain a significant number of pixels). Out of all the significant blobs found, the algorithm picks the one with the largest area (number of pixels) as the largest *light* region.

Next, the raw image is thresholded by the mid-point between the *light* and *medium* pixel cluster centroids. This time, all pixels below the threshold are considered logical 1 and all above are logical 0. This new binary image is combined with E^+ using a logical pixel-wise *and*. The resulting binary image contains *edge structure* in the *dark* regions of the raw image.

The *edge structure* is applied to the connectivity analyses algorithm to find all significant connected edge structures in dark regions of the raw image. The resulting processing region is then determined to be the largest *edge structure* in a *dark* region which is within the largest *light* region. If no processing region is found, then the largest *edge structure* in a *dark* region becomes the processing region. And if there where no significant *edge structures* in a dark region found a warning message is issued and the entire image is used as a processing region.

References

[1] R.G.Nornholm, "Space Robotics", *Internal Technical Report MDSSC-SSD*, Report No. MDC 91Q0804-12.

[2] *MDSSC-SSD Maintainability Input for CCO-108 (SS Robotic Costing Proposal)*, Contract NAS 9-18200, December 1992.

[3] W.F. Fisher and C.R. Price, *"Space Station Freedom External Maintenance Task Team"*, Final Report, NASA Johnson Space Center, July 1990.

[4] D.H.Kite and M.Magee, "Determining the 3D Position and Orientation of a Robot Camera using 2D Monocular Vision", *Pattern Recognition*, Vol. 23, No. 8, pp. 819-831, 1990.

[5] R.C.Gonzalez and P.Wintz, *Digital Image Processing*, 2nd ed., Addison Wesley, 1987.

[6] R.Y.Wong and E.L.Hall, "Scene Matching with Invariant Moments", *Computer Graphics and Image Processing*, Vol. 8, pp. 16-24, 1978.

[7] M-K Hu, "Visual Pattern Recognition by Moment Invariants", *IRE Transaction on Information Theory*, Vol. IT-8, pp. 179-187, 1962.

[8] Vishal Markandey and Rui J.P. deFigueiredo, "Robot Sensing Techniques Based on High-Dimensional Moment Invariants and Tensors", *IEEE Transactions on Robotics and Automation*, Vol. 8, pp. 186-195, April 1992.

[9] Y. Kuno, Y. Okamoto and S. Okada, "Robot Vision Using a Feature Search Strategy Generated from a 3-D Object Model", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13, No. 10, 1991.

[10] D. G. Lowe, "Fitting Parameterized Three-Dimensional Models to Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13, No. 5, 1991.

[11] R. P. Johnson, "Contrast Based Edge Detection",*Pattern Recognition*, Vol. 23, No. 3/4,, pp. 311-318, 1990.

[12] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York, 1981.

[13] B. K. P. Horn, *Robot Vision*, The MIT Press, Cambridge, Massachusetts, 1986.

AIAA-94-1256-CP

GA-OPTIMIZATION FOR RAPID PROTOTYPE SYSTEM DEMONSTRATION

Jinwoo Kim and Bernard P. Zeigler

Artificial Intelligence and Simulation Research Group
Department of Electrical and Computer Engineering
University of Arizona, Tucson, Arizona 85721
jwkim@helios.ece.arizona.edu

Abstract

This paper discusses an application of the *Genetic Algorithm*, a parallel and global search technique that emulates natural genetic operations. Real application problems often require optimization of a large number of parameters with high precision. Since the existing Genetic Algorithms do not represent the parameter sensitivities, we have devised a novel scheme of *Hierarchical Genetic Algorithm* to solve complicated engineering problems. Using this approach, the higher level GAs propose promising search spaces, while the lower level GAs search in more detail with additional parameter sets. This decreases the complexity of search and utilizes the computing resources efficiently. This scheme has been used to design an autonomous control systems for space-based resource processing plants.

1. Introduction

Applications of computer technology are expanding from pure data processing to information and knowledge processing which enables Computer-Aided System Design. Knowledge-based system applications are characterized by symbolic processing, nondeterministic computation, dynamic execution, high potential for parallel and distributed processing and knowledge management. However, fundamental physical limits of current technology have not been overcome for the more sophisticated computation-intensive problems, such as predictive modeling and forecasting, design automation, large scale, simulation and artificial intelligence. The combination of technology and economic factors make parallel and distributed computing systems attractive and effective for a large variety of intelligent machine applications [9].

The emergence of massively parallel computers has also fueled a growing interest in problem solving systems based on principles of evolution and heredity. One class of such *evolution strategies* is Genetic Algorithms [14]. The remarkable success demonstrated by Genetic Algorithms (GA) in search, optimization and learning has substantially increased interest in their potential application to modeling, simulation and design of complex real world systems [1, 6]. Such applications include identification and calibration in model construction and subsequent model-based control synthesis and policy optimization. However, complex simulations typically require large execution times to evaluate alternatives, or in GA terms, to obtain fitness values for newly generated chromosomal individuals. Such lengthy simulations present a major bottleneck to GA application since tens, or even hundreds, of individuals may need to be evaluated in every generation. Parallel processing offers promise of reducing this bottleneck along two mutually supporting avenues: 1) speeding up the simulation needed to estimate fitnesses using distributed simulation methods [5] and 2) parallelizing the evaluation and processing of fitness information. Both avenues are under active investigation and indeed a computer architecture to support their integration has been suggested [19].

Real application problems often require optimization of a large number of parameters with high precision. These parameters increase the complexity of the search problem. In existing approaches, a chromosome representing the parameters does not contain information about their sensitivity, even though parameters influence the system performance to different degrees.

We have developed a novel scheme of a Hierarchical GA optimizer which executes multiple GA modules to solve complicated problems. These GA modules are constructed hierarchically and creation/deletion is performed dynamically based on the performance of each module.

Each GA module deals with a different degree of abstracted models for evaluation and a different number of parameters for optimization. High level GA modules usually search for fewer parameters which are more sen-

Copyright ©1993 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

sitive to the system performance. They are looking for milestones of promising search region instead of accurate solutions. The candidate individual selected from the high level GA module represents a sub search space and they are sent to the lower level GA modules for more detail search. The lower level GA takes advantage of the received information, and employs a greater number of parameters for further optimization.

The solutions found at the lower level GA module are reported back to the higher level GA module, if it is better than the candidate individual from parent module. This information is used to update the fitness of the parent. As the purpose of high level GA module is not to find actual solution, the models of this level are not necessarily accurate. In order to speed up GA search, the high level GA modules access less accurate models which can reduce simulation-based fitness evaluation time. The basic concept is that of successive approximation provided by a nested sequence of models [13].

2. Hierarchical Genetic Algorithms for Complex Problems

A simulation model of such a complex architecture is most naturally formulated as a variable structure model [21]. A Hierarchical GA is implemented on a self-organizing variable structure, where creation/deletion of modules are determined by their performance.

2.1 Brief Review of Asynchronous Genetic Algorithm

The GA (genetic algorithm) is a probabilistic algorithm which maintains a population of individuals, $P(t) = x_1(t), \ldots, x_n(t)$ for iteration t. Each individual represents a potential solution to the problem at hand, and, in any evolution program, is implemented as some (possibly complex) data structure S. Each solution $x_i(t)$ is evaluated to give some measure of its *fitness*. Then new population (iteration $t+1$) is formed by selecting the more fit individuals (select step). Some members of new population undergo transformation (recombine step) by means of "genetic" operators to form new solutions. There are unary transformation m_i (mutation type), which create new individuals by a small change in a single individual ($m : S \to S$) and higher order transformations c_j (crossover type), which create new individuals by combining parts from several (two or more) individuals [14]. The control parameters for genetic operators (probability of crossover and mutation) need to be carefully selected to achieve acceptable performance [15]. After some number of generations the program converges and is successful if the best individual represents the optimum solution.

We have developed concepts for parallel genetic algorithms that are especially oriented to simulation-

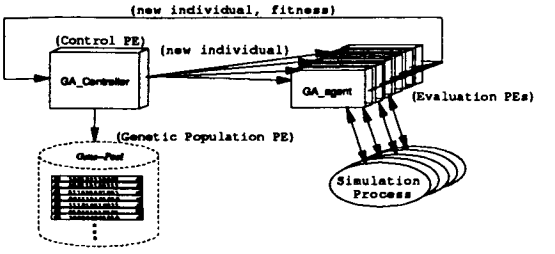


Figure 1: The Architecture for Asynchronous Genetic Algorithm Simulation

evaluated individuals on high performance computers. We have investigated a class of *Asynchronous Genetic Algorithms* (AGAs) which does not need to be synchronized by generations to create successive populations. In a multiprocessor architecture, individuals are evaluated concurrently and a central agent updates the genetic population continuously as the evaluation results become known. The motivation behind such asynchronous updating is the recognition that not only may simulation runs be time consuming, but their completion times may be highly variable. This variability is quite common in performance measuring simulations [1]. When such variability is significant, the barrier synchronization imposed by conventional GAs can greatly impede search progress since it requires that processing cannot proceed to the next generation until the slowest individual in the current population has completed its evaluation [7]. In contrast, the AGA allows new individuals to be tested as soon as both the information and the computer resources are available to do so.

A concern immediately raised in the AGA paradigm is that the blending of generations occasioned by such asynchronized processing might adversely affect the recombination schemes underlying GA search. Certainly, the supporting theory typically limits selection, mating, crossover and other operations to members of the same generation [6, 8]. Fortunately, some results in the literature suggest that search time and search success are not degraded, at least in application to typical test function suites [4]. We note that such artificial fitness functions do not include time dependence that might appropriately suggest the necessity for generational integrity.

As shown in figure 1, the processing elements (PE) in the asynchronous genetic algorithm can be categorized as : *genetic population PE, evaluation PEs* and *control PE*. While the PGA executes serial GAs in the

[1] It certainly occurs when runs are executed in conditional mode where termination depends on pre-established criteria (e.g., a run may be terminated as soon as failure to achieve a prescribed goal is obvious). However, run time variability may also arise when runs span a fixed observation interval (on the model time base). This may be due to the variability in workload encountered by the simulation engine or in the resources allocated to the particular trial

multiple processing elements with subpopulations, the asynchronous genetic algorithm evaluates a single individual in a processing elements (*evaluation PE*) at a time. The total population is always contained in the *genetic population PE* which executes genetic operations and updates the population. It also generates new individuals whenever it receives request from the *control PE*.

Message transfer between the *genetic population PE* and the *evaluation PE* is controlled by the *control PE* : it delivers evaluated individuals to the *genetic population PE* and requests new individuals for the *evaluation PE*. Thus the *evaluation PEs* keep evaluating individuals with which the *genetic population PE* continuously updates the population.

2.2 Resolution Increasing Scheme in the Hierarchical Genetic Algorithm

A binary chromosome, a unique knowledge representation scheme of Genetic Algorithms, provides a way of controlling the accuracy of parameters. The size of the binary code determines the number of points to be investigated. As more bits are employed, the search points increase dramatically (exponentially). Therefore, longer string size may provide accurate parameter values, but it also makes the GA to search through a large number of points.

As shown in figure 2, same size of binary code can increase accuracy as search space changes. At level 1, the original search space is defined by MIN and MAX. We employs 3 bits and there are 8 possible search points. If a certain point (binary code) is selected, the distance between its neighbors becomes a new sub search space. Therefore the selected candidate individual at the high level GA module has meaning as representation of its neighbors (sub search space) rather than an actual value itself. The same size of binary code is employed with the new search space, which increases the accuracy of the parameter value.

2.3 Expanding Search Parameters in the Hierarchical Genetic Algorithm

The previous section explains how search accuracy is controlled by the Hierarchical GA. If the search problems involve a large number of parameters, the GA takes longer or directs to local minima. The Hierarchical GA employs an expanding search parameter scheme. The higher level starts to search for a small number of parameters. The result obtained at the higher level is sent to the lower level GA, where extra parameters are included to the received parameters. The lower level GA takes advantage of the received information so that it need not search all the parameters from the beginning.

The expanding parameter scheme in the Hierarchical

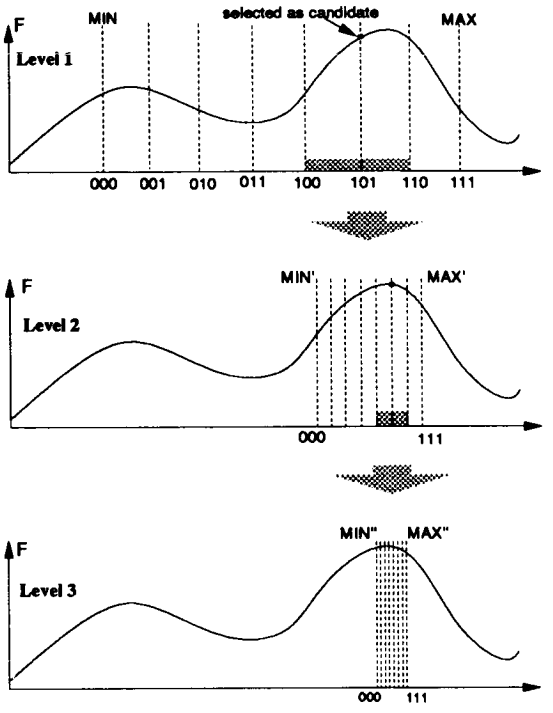


Figure 2: Resolution Control Scheme in Hierarchical Genetic Algorithms

GA also helps determine the optimal number of parameter sets. Complicated designing problems involve the evaluation of a large number of parameters, where the type of parameters as well as their appropriate values are often unknown.

2.4 Execution of Multiple GA Modules in Parallel

The selected individuals during the search operation from the root AGA create lower level GA modules as shown in figure 3. The time taken by the module for checking its population and selecting a candidate individual may be either deterministic or nondeterministic. In this experiment, we choose a variable selection interval scheme, in which the time intervals of subsequent checking become larger as the GA search continues. This strategy is based on a characteristic of GA search, the fitness of population increases faster in early search stages and saturates to a certain level. Smaller checking intervals enables the GA to choose new candidates before stagnation.

When a lower level AGA module selects an individual as a candidate for the next level, it also reports fitness to the parent AGA module. This feedback information updates the fitness of parent individual. But the individual structure of the parent and child may not be comparable to each other because they represent different parameter sets or different search space. The popu-

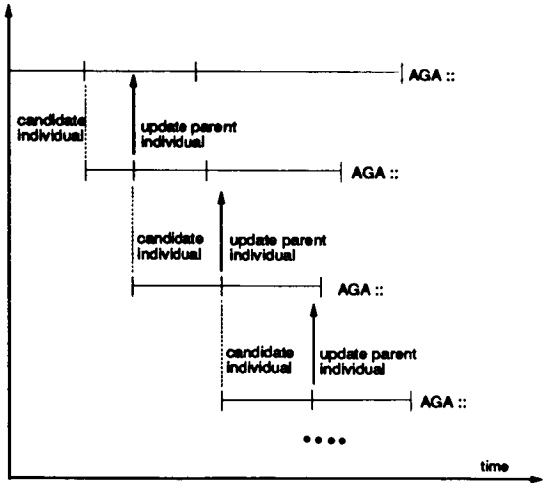


Figure 3: Creation and Execution of multiple AGA in Self-Organizing Hierarchical GA

lation of child AGA searches through the space which is suggested by a candidate individual from parent AGA. The fitness of the lower level individual, if better, updates the fitness of parent individual. If updated, the fitness of parent no longer represents an actual fitness of the binary code. But, since the fitness is increased by a certain amount, its value can not be represented in its environment. The individual now has a higher fitness and has a greater probability of being selected for the next evaluation.

Figure 4 shows the module components at each level. Each module contains a *controller* which executes *AGAs* to solve a given problem and select candidate individual(s) reported from AGAs. It creates/deletes lower level modules and communicates with higher/lower level modules. This *controller* is the intelligent component of module and is implemented by a rule-based expert system. The module is defined as a class in an object-oriented programming environment (Chez-Scheme [3]) and the same module structure is created by higher level object. This module is program-interfaced to an AGA procedure written in C. The decision making component is implemented in a symbolic processing language, while the numeric computation procedure is written in C.

When the module receives a problem, it may expand search parameters as well as increase resolution. With any given problem, the controller first expands parameters and sends them to the PARA-AGA for GA search. The selected individuals at the PARA-AGA are sent to HIGH-AGA to increase resolution by the scheme explained above. The fitness of the selected individual at the PARA-AGA are also reported to the high level HIGH-AGA to update the fitness of parent individual. The candidate individuals for the next level are selected at the HIGH-AGA where the selected individual is also

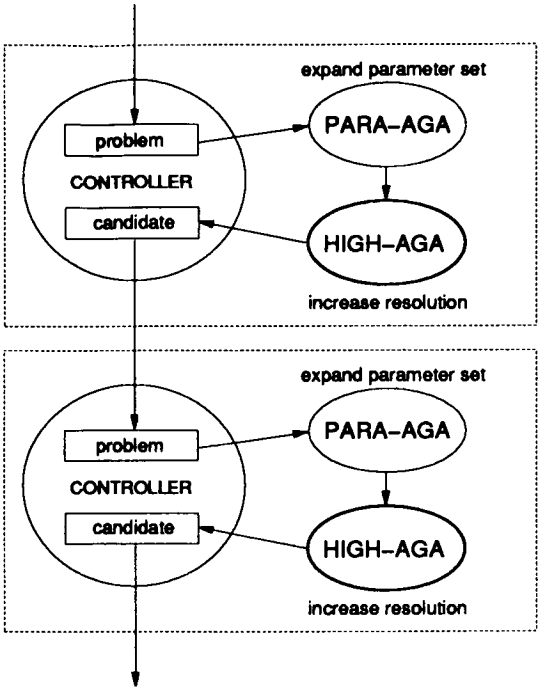


Figure 4: Module Components in the Self-Organizing Hierarchical GA

reported to the PARA-AGA of same module.

3. Design of Control System using Self-Organizing Hierarchical GA Environment

A working prototype of a plant for producing oxygen from Martian atmosphere, is constructed at NASA-UA Space Engineering Center [16]. The purpose is to evaluate the best designs and operation parameters for the Mars mission. Martian CO_2-rich atmosphere is filtered and compressed to a temperature and pressure suitable for electrocatalysis in a Zirconia-based oxygen cell. Design issues include the size of the inlet pipe, power requirements of the compressor and design of the oxygen cell including: cell configuration, material properties, electrical parameters such as operating voltage and current density, electrode materials, and method of application

In this experiment, we try design an optimal FLC to control the temperature of the oxygen production system. The basic idea of the fuzzy control centers around the *labeling* process, in which the reading of a sensor is translated into a label as done by human expert controllers [12]. With expert supplied membership functions for this labels, a reading of a sensor can be *fuzzified* and *defuzzified*. It is important to note that the transition between labels are not abrupt and a given reading might belong to several label region.

The fuzzification and defuzzification processing does not need to be sequential. The input signal can be

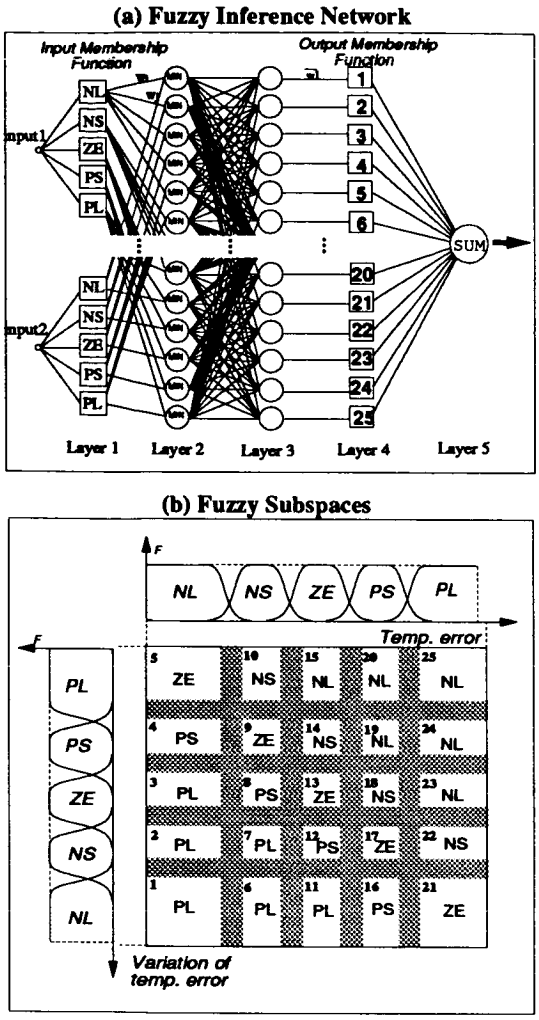


Figure 5: Fuzzy inference network and fuzzy subspaces: (PL:Positive Large) (PS:Positive Small) (ZE:Zero)(NS:Negative Small) (NL:Negative Large)

fuzzified/defuzzified simulatenously by matching membership functions. Therefore fuzzy control processing can be adapted to a parallel neural network structure where each *neuron* represents functions (fuzzy membership) and *links* represent the weight of a fuzzy rule.

Figure 5(a) shows the structure of the Fuzzy Neural Net Control System (FNC) and its fuzzy subspace (Figure 5(b)) [10]. In this experiment, 5 input membership functions are assigned to each input signal and 5 output membership functions are used to compute fuzzy output signal.

While an earlier Fuzzy Logic Controller [16, 20] was implemented in rule-based form (*if-then*), the FNC employs a parallel inferencing network structure. Due to the parallel fuzzification/defuzzification scheme, the FNC can improve real-time performance of the control system for practical application.

The performance of the FNC is determined by the

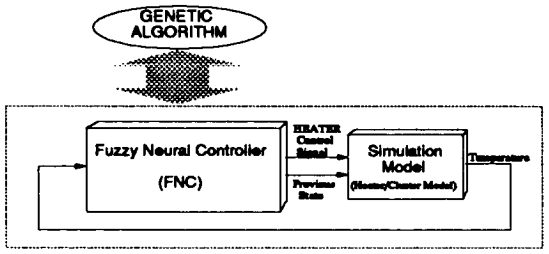


Figure 6: GA optimization of the FNC module

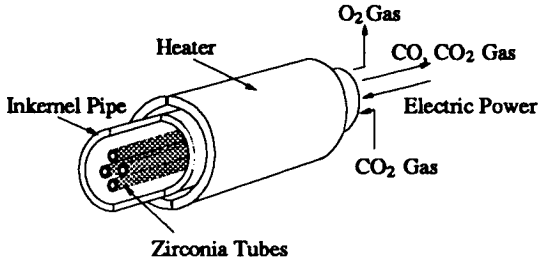


Figure 7: Oxygen Production System Cluster

input membership functions of layer 1 which fuzzify the input signals and the output membership functions of layer 4 which defuzzify normalized firing strengths. A membership function is specified by number of parameters.

In order to find a high performance fuzzy membership functions without the help of human expertise, it is necessary to employ computer-aided optimization. Since tuning the membership functions requires adjusting many parameters simultaneously, hill-climbing search methods would suffer from the complexity of the search space.

For this reason, a probabilistic optimization method utilizing evolution strategies, such as Genetic Algorithm (GA), was employed to find optimal membership functions. Since optimizing multi-parameter problems takes a long time, we developed new form of GA which is especially oriented to parallel computers that can satisfy the real-time constraints of the system.

Figure 6 shows the interaction of the FNC, simulation model and GA-optimizer. The FNC operates the simulation model, such as heater/cluster model of the Mars Oxygen Production System (OPS).

The OPS, shown as in figure 7, includes Zirconia tubes located symmetrically inside a cylinder. A radiation heater is wrapped around the outer surface. With this configuration, the majority of heat transfer between the outer surface and the oxygen gas inside the system is due to radiation. Applying the one-dimensional heat equation with lumped temperature distributions for the surface and oxygen temperatures we obtained two first order differential equations as provided below. The T_p

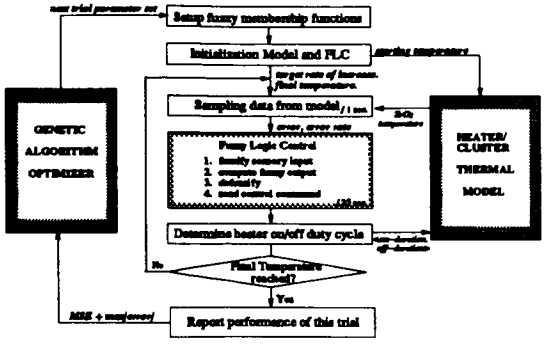


Figure 8: Individual evaluation procedure:FNC operation with heater/cluster thermal model

represents the pipe temperature and T_z is Zirconia tube temperature. A variable SW is either 0 or 1 to control the heat source (heater). The objective of the FNC is to increase the temperature of the Zirconia tubes at a constant rate until a goal temperature is reached [17].

$$\frac{dTp}{dt} = 2.75 \times SW - 4.42 \times 10^{-12}(T_p^4 - T_z^4) - 8.65 \times 10^{-4}(T_p - 278.0)$$

$$\frac{dTz}{dt} = 4.42 \times 10^{-12}(T_p^4 - T_z^4)$$

As shown in figure 5, there are two input signals to the FNC e.g., temperature increase error rate (input1) and rate of its error rate (input2). Based on two inputs, the FNC produces an output command which controls on/off duty cycle of the heater element in the model. As shown in figure 5, we employed 5 membership functions for each input signal and 5 membership functions for the output signal.

Figure 8 provides detailed procedures of the FNC integrated with the GA-optimizer. An individual of a GA represents one trial set of fuzzy membership functions. The GA optimizer sends a parameter assignment to the FNC which determines its fuzzy membership functions. The model is reset to its initial conditions (starting temperatures). The operational specifications such as desired temperature increase rate and goal temperature are set inside the controller. The performance of a trial individual fitness is measured as the sum of the MSE (Mean Square Error) between actual temperature increase rate and desired one and maximum absolute value of error of temperature increase rate.

3.1 Design of the FNC for the Oxygen Production System

Our primary objective is to design optimal fuzzy membership functions that perform well with given operational specifications while utilizing minimal human expertise. The controller increases the temperature of the cell at a constant rate.

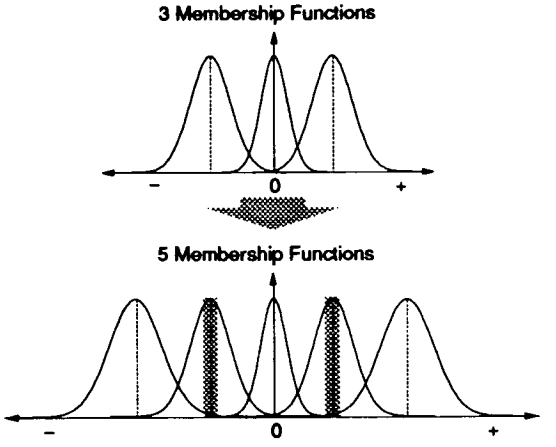


Figure 9: Expanding Fuzzy Membership Functions

Designing an optimal FLC involves the investigation of several alternatives, such as type of membership functions and the number required. A single-level GA starts to optimize the FLC based on the assumption that a given FLC specification, such as type or number of membership functions, is optimal. But real world application problem is often too complicated to determine the correct system specification.

Hierarchical GA solves this problem by changing its structure according to the performance of each module which employs a different number of fuzzy membership functions and parameter resolution. Starting from a small number of parameters, it expands search parameters and their resolution as they create lower levels. The lower levels take advantage of information found at the upper level AGA module.

Figure 9 shows how search parameters are expanded as Hierarchical GA creates lower level modules in the example of designing a FLC. The upper level module starts to design the FLC with a small number of membership functions. Designing a FLC with fewer membership functions is relatively easy compared to a large number of membership functions. Even though the upper level need not find the best membership function, it does provide some information to the lower level which supports the design of an optimal FLC. Since the membership functions found at the upper level are optimized based on constraints of a small number of parameters, the lower level GA modules give small tolerances to the received parameters. This is due to the effect of new parameters on the old parameters optimized earlier. Figure 9 illustrates how to expand membership functions, the shade area of membership function represents its tolerance.

As we increase the number of employed fuzzy membership functions, the fuzzy rule table must also be expanded. Figure 10 shows ways of adding more slots to

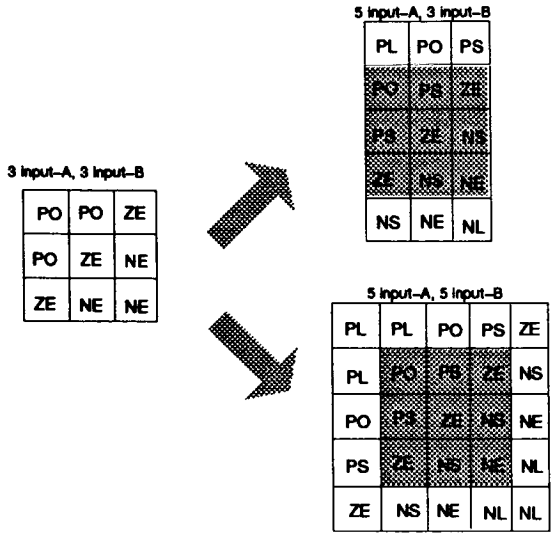


Figure 10: Expanding Fuzzy Rule Table

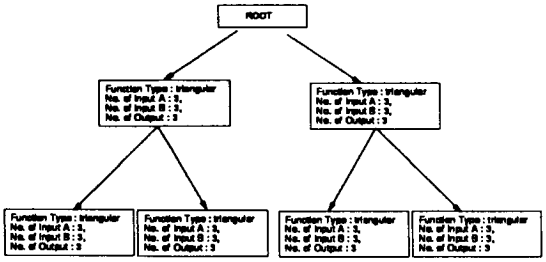


Figure 11: Tree of Various FLC Specifications

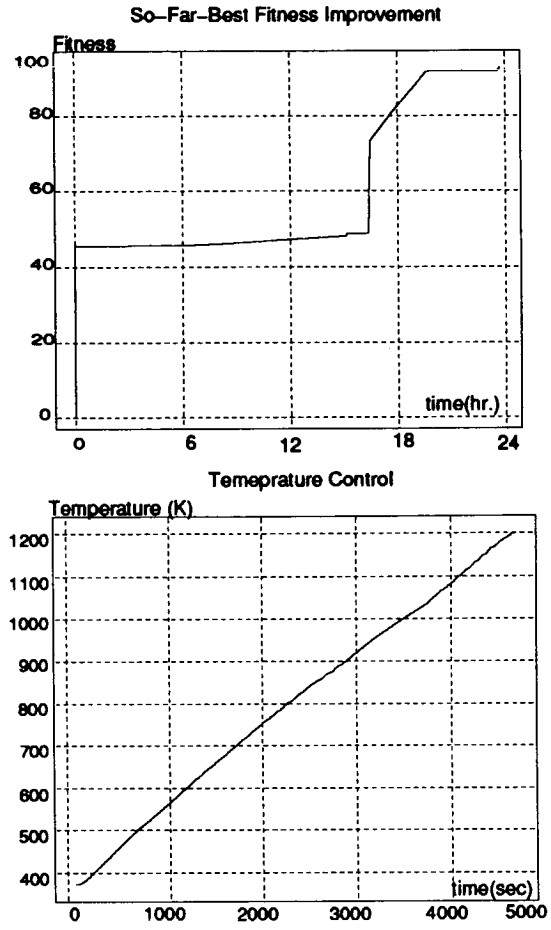


Figure 12: Simulation Results from Self-Organizing Hierarchical GA to design optimal FLC

the fuzzy rule tables. Since the fuzzy rules are optimized based on constraints of fewer membership functions (for example, 3 input A,B, and 3 output membership functions), the expanded fuzzy rules need to be optimized with not only more slots but also some degree of tolerance of suggested rule parameters.

Figure 11 shows a tree of various specifications of the FLC in which the Hierarchical GA searches through optimal design. Hierarchical GA first optimizes fuzzy rules which are more sensitive to the FLC performance at the root module. The small number of fuzzy membership functions with small parameter variance were used in order to maximize the sensitivity of fuzzy rules. The fuzzy rules found at the root module are sent to lower levels, where two different membership function types, such as triangular and bell shape are employed. The lower levels have wider search ranges in the parameters and utilize the fuzzy rule information received from the root. A greater number of fuzzy membership functions are employed when the lower level GA modules are created.

Figure 12 shows the simulation results that illustrate how the Hierarchical GA investigates various FLC specifications to design an optimal controller. The fitness improvement of Hierarchical GA shows that when a certain GA module is executed, the fitness increases suddenly. The FLC specifications of the module provides the best performance among other FLC specifications. The population of each module represents different species, because they expresses different number of parameters and search spaces. When a certain species (the correct one) is created, the performance of the Hierarchical GA improves in a step like manner. The temperature profile shown in the figure 12 is that of by the suggested optimal FLC in the Hierarchical GA.

4. Conclusions

Real world application problems often require optimization of a large number of parameters with high precision. These parameters increase the complexity of the search problem. In existing GA, a chromosome representing the parameters does not contain information about their sensitivity, even though they influence the system performance to different degrees.

We have devised a novel scheme of Hierarchical Genetic Algorithms in self-organizing variable structure

environment for complex real world problems. The design of a temperature control system for the oxygen production plant was selected as an experiment. Since conventional control schemes are limited their functionality to relatively simple applications, Fuzzy Logic/Neural Net control methods are received more attention for the sophisticated applications. The parameters embedded in the controller need to be optimized for the required control performance.

In this paper, a Hierarchical GA investigates various FLC specifications using variable structure simulation. More sensitive parameters, such as fuzzy rules, are optimized before other parameters. Higher level GAs search for candidate individuals that might contain the optimum in a given search space. These candidates are sent to the lower level to be investigated in greater detail. If better solutions are found at a lower level, they are reported back to the higher level and incorporated into its on-going GA search. The higher level GAs search in a sparse space with fewer parameters that influence the system performance significantly. In order to reduce GA search time, higher levels also utilize less accurate models for which simulation-based evaluation time is reduced.

The simulation exhibited interesting search behavior : when a good GA module is discovered, the performance increases suddenly. This suggests that not only has a good design been found, but that all other design frameworks can be eliminated.

References

[1] Back, T. and Schwefel, H-P.,"An Overview of Evolutionary Algorithms for Parameter Optimization", *Evolutionary Computation*, Vol.1, No.1, MIT Press, 1993.

[2] Collins, R. J. and Jefferson, D. R., "Selection in Massively Parallel Genetic Algorithms", Proceedings of the 4th ICGA, Morgan Kaufman, San Mateo, CA, 1991.

[3] Daniel, S. M., "Motorola Software Tools for Chez Scheme : User's Guide and Reference Manual", Motorola inc. Scottsdale AZ, 1991.

[4] DeJong, K. *An Analysis of the Behavior of a Class of Genetic Adaptive System*, Ph.D Dissertation, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, 1975.

[5] Fuji, R. M., "Parallel Discrete Event Simulation", Communication of the ACM, Vol.33, No.10, Oct. 1990.

[6] Goldberg, D. E., "Genetic algorithms in search, optimization, and machine learning", Addison-Wesley, Reading, MA, 1989.

[7] Grefenstette, J.J., "Parallel Adaptive Algorithms for function optimization", Tech. Report No. CS-81-19, Vanderbilt University, Computer Science Department, Nashville. 1981.

[8] Holland, J., "Adaptation in Natural and Artificial System", University of Michigan Press, Ann Arbor, 1975.

[9] Hwang, K. and Degroot, D., "Parallel Processing for Supercomputer and Artificial Intelligence", McGraw-Hill, New York, 1988.

[10] Jang, J.S., "ANFIS:Adaptive-network-based fuzzy inference systems" *IEEE trans. on System, Man, and Cybernetics*, 1992

[11] Kim, J. and Zeigler, B. P., "Designing Fuzzy Neural Net Controllers using GA Optimization", accepted to International Joint Symposium IEEE/IFAC on Computer-Aided Control System Design. 1994.

[12] Lee, C. C., "Fuzzy logic in control systems : fuzzy logic controller-part 1.". *IEEE trans. on System, Man, and Cybernetics*, 20(2):404-418, 1990

[13] Maumov, Y. and Meystel, A., "Optimum Design of Multiresolutional Hierarchical Control System", Proceedings of the 7th International Conference of Intelligent Control. Glasgow, UK. 1992.

[14] Miachalewicz, Z., *Genetic Algorithm + Data Structure = Evolution Programming*, Springer-Verlag, New York, 1992

[15] Schaffer, J., Caruana, R., Eshelman, L. and Das, R., "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization", *The Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publisher, Los Altos, CA, 1989

[16] Schooley L., Zeigler B., Cellier F., Marefat M., Wang F., *"High Autonomy Control of Space Resource Processing Plants"*, Control Systems Magazine, June, 1993

[17] Stephens, W. P, Cummings, D and Vincent, T.L. "Automation of a Nonmechanical Oxygen Compressor", Progress Report in the Dept. of Aerospace and Mechanical Eng. University of Arizona, Nov. 1, 1991.

[18] Zeigler, B.P., *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*, San Diego, CA. Academic press, 1990.

[19] Zeigler, B. P. and Louri, A. "A Simulation Environment for Intelligent Machine Architecture", Journal of Parallel and Distributed Computing, No. 18, 1993.

[20] Zeigler, B.P. and Kim, J., "Event-based Fuzzy Logic Control System", *Proceedings of 8th IEEE International Symposium on Intelligent Control*, Columbus, OH, 1993.

[21] Zeigler, B.P., Kim, T.G., Lee, C., "Variable Structure Modelling Methodology : An Adaptive Computer Architecture Example", Transactions of The Society for Computer Simulation, Vol. 7, No. 4, 1991.

A ROBOT CONTROL FORMALISM BASED ON AN INFORMATION QUALITY CONCEPT

A. Ekman, A. Törne, D. Strömberg
Dept. of Computer and Information Science, Linköping University
FOA (National Defence Research Establishment)
Linköping, Sweden

Abstract

The efficiency of an autonomous robot navigating in indoor environments depends crucially on the ability of the robot to exploit spatial relationships extracted from perceptions of its environment. Smith, Self, and Cheeseman[23] describes a formalism based on Kalman filter theory, where perceptions from different locations can be combined to improve the accuracy of the robot pose estimate. We argue that while accuracy is an important property of perceptions of the robot state, a more important property of perceptions of the environmental state are their temporal and spatial range of applicability, which will be referred to as perceptual relevance. This paper introduces a relevance measure based on Jaynes maximum entropy principle, measuring the relevance of a spatial description of the robot environment. The conjunction of accuracy and relevance is denoted information quality. A formalism based on the information quality concept is developed for the class of one-agent applications, for which the formalization of the dependency between perceptions and actions of a robot is straightforward.

1 Introduction

A robot can be viewed as a controller, the purpose of which is to transform the current system state into a goal state. After having executed the action sequence, the system state should be closer to a goal state. If we by "world" denote the conjunction of robot and system, this paper is based upon that essentially three issues determine the performance of the state transformation process: 1) The accuracy and relevance of the robot's perception of the world state, 2) The robot's capability to find an action sequence that forces the current model state into a desired goal state, and 3) The precision of the transformation from abstract to physical actions.

The behavior of the robot corresponds to what actions the robot selects to execute in a particular situation. For sensorless robots, behavioral information in the form of action sequences are given a priori, and may not change due to external events during operation. Although this is a straightforward way to implement robot behavior, the robot requires a well-defined working environment where the properties of each object must be accurately specified. In effect, all information is given to the robot a priori, and a major problem is to maintain a configuration of the working environment that is consistent with the specification.

More flexibility is achieved if the robot is capable of acquiring information about the true configuration of the working environment during operation. Robots capable of acquiring information during operation may be classified as being either reactive[1,2] or deliberate. While reactive behavior commonly is hard-wired into the robot, deliberate behavior is exhibited by robots maintaining an explicit world model.

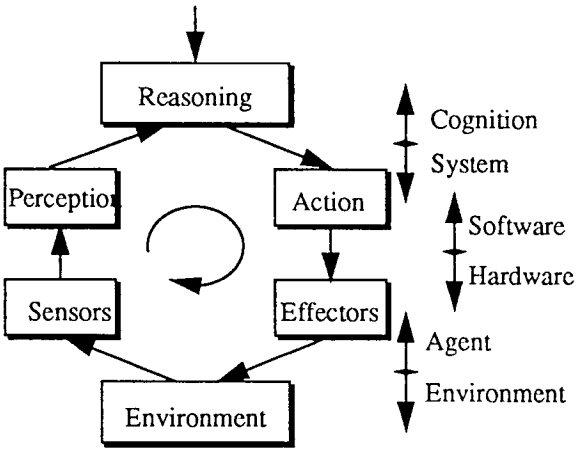


Fig. 1. Structure of a deliberate robot

Obviously, reactive robots are inherently autonomous, while deliberately behaving robots may be anything from autonomous to tele-operated. In the autonomous case the explicit model is implemented in the robot software, while in the tele-operated case the model exists in the mind of the expert controlling the robot. Issues affecting the performance of a deliberatively behaving robot will be addressed in this paper. Throughout the paper, except where explicitly stated, the only assumption being made

Copyright © 1993 by Anders Ekman. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

concerning the deliberately behaving robot is that it possesses an explicit world model.

In *Fig.1*, a useful way to describe the structure of a deliberate robot is presented. By splitting the graph horizontally at increasing heights, one gets the following sequence of dichotomizations: environment/agent, hardware/software, system/cognition. The figure also illustrates the cyclic processing of perceive-reason-act which starts when a task has been given.

A distinguishing feature among deliberate robots is the degree of human interaction, spanning from autonomy to tele-operation. For an autonomous robot the "reasoning" module (see *Fig.1*) corresponds to a computer program while for a tele-operated robot the it corresponds to the human operator. Albeit having this difference, all deliberate robots need high quality information in order to do proper inference. By keeping the information quality above some pre-defined level, the likelihood of erroneous inference is kept sufficiently low. In this paper we will develop means for preserving the information quality, which are applicable to a large class of deliberate robots.

The formalism developed in this paper is based on an assumption of a one-agent application. The formalism might however be extended to cover many-agent applications as well. One distinguishing feature between one- and many-agent applications is that while failure to execute a plan in a one-agent application is caused either by poor information or by poor control, for many-agent applications an additional cause of plan execution failure are actions executed by other agents. Without doing any further elaboration on the class of many-agent applications, it suffices to notice that more powerful models must be developed and that real-time constraints become crucial[16].

The robot uses the world model for interpreting the present situation. Therefore, it is of great importance that the model discrepancy is small. On the other hand, a too detailed model suffers from high time and space complexity. The difficulty to satisfy these somewhat contradictory constraints is one reason why many deliberate robots are either too slow or too error-prone. At the core of the problem are the issues of uncertainty and complexity. Typically, reducing the complexity of a model increases the uncertainty and vice versa. However, by using application-specific heuristics, it is possible to suppress world properties of minor importance, thereby simplifying the model. In this way a model with both low discrepancy and moderate complexity can be establish. For example, in the mobile robot case, a 2D (global) map suffice for robot navigation, while a 3D (local) map is needed for many object manipulation tasks. By using this heuristics, a 2D/3D composite model is created, with a fair trade-off between complexity and discrepancy. The problems to represent and reason under uncertainty have been addressed in several papers[10, 11, 12, 25]. Although this is an important research area, the work do normally not consider the problem of uncertainty and relevance maintenance, in particular not when the uncertainty and relevance varies in time and space.

Having developed an appropriate model, the next question is why and when the model should be updated with fresh information? In order to answer the first question, we recapitulate that the model should have limited space and time complexity. This inevitably leads to an information loss. Accordingly, situations may occur where ignorance of some world property will result in robot malfunction, although such situations may be very rare. Crucial for the prevention of robot malfunction is to maintain a low model discrepancy, since the next action to execute is determined by the model interpretation. Obviously, the penalty for having a model of low complexity is that it must be updated frequently to keep the discrepancy low.

When to update the model depends on how low one want to keep the likelihood of robot malfunction. While it is obvious that a high model discrepancy increases the risk of robot malfunction, it is very difficult to calculate the probability of robot malfunction as a function of the model discrepancy. The reason for this is that in order to calculate the discrepancy the model state must be compared to the world state, which contains an infinite number of elements. An approach to this problem is to introduce a threshold value for each perception, representing the minimum permitted information quality of the perception. After each executed action during the execution of an action sequence, the robot checks that each perception has a quality exceeding the corresponding information quality threshold. If this is not the case, a sensing action is executed to increase the information quality of the perceptions. Otherwise, the next action in the sequence is executed.

2 Model - world duality

The correspondence between the model and the real-world is established through the following definition.

Definition 2-1 To find a solution to a real-world problem is analogous to the abstract problem of finding a path, p, satisfying the predicate Q(p), from an initial state to a goal state in a model state space, S.

For instance, Q(.) might represent the predicate "shortest(.)", "least expensive(.)", or "most safe(.)".

Although *Def. 2-1* is rather general, a straightforward interpretation within the framework of this paper is possible. The problem space, S, corresponds to the space of possible model states. The path, p, corresponds to a sequence of actions, while Q(t) is interpreted as "Information Quality high enough along the path".

Ideally, the abstract plan is in perfect agreement with the solution to the real-world problem. In practice, though, this is rarely the case and the robot must perform plan validation iteratively during the execution. The efficiency in the detection of plan failures and the subsequent re-planning is closely related to the performance of the robot, and could accordingly be taken as a measure of the same.

Each cycle of processing in *Fig.1* corresponds to the transition between two states in the problem space - ideally towards a goal state.

This duality between the cyclic processing of the physical system and the problem space transitions suggests the possibility to analyze plans in the problem space before executing them on the physical system.

3 Concepts

One-agent applications

A large class of robotic applications are one-agent applications. In a one-agent application, it is assumed that all changes to the system state are caused by the actions of the sole agent. Consequently, for one-agent applications it is straightforward to formalize the dependency between actions and perceptions of an agent.

World

The world is composed of two entities, agent and environment. This is in accordance with the one-agent application assumption. The agent may correspond to either an autonomous or a tele-operated robot.

Sensing

In order to gather information about the world, the agent must use sensors. Sensors measure either the state of the agent or the state of the environment. Sensing, σ, maps the world state to perceptions, which in turn can be mapped to a more abstract perception, or be combined with previous perceptions to give a more accurate perception.

Perception

Perceptions provide descriptions of details of the world. To distinguish between perceptions with different properties, perceptual classes are introduced. Within each perceptual class, perceptions are distinguished by their creation time. Accordingly, a perception from perceptual class i, created at time k will be denoted $p_{i,k}$. Perceptions acquired by the agent up to time k, where $k \in N$, is represented by the perception vector P_k

$$P_k = (p_1, p_2, \ldots, p_r) \tag{1}$$

where r is the number of perceptual classes and p_i corresponds to a set of acquired perceptions of the i:th perceptual class, that is

$$p_i = \{p_{c,k-k_1}, p_{c,k-k_2}, \cdots, p_{c,k-k_e}\} \tag{2}$$

where $0 \le k_e < \ldots < k_2 < k_1 \le k$.

Perceptions are created either by using data from one or more sensors or by combining previous perceptions into a more abstract or more accurate perception. To create more abstract perceptions, feature extraction algorithms are applied, while to create more accurate perceptions spatio-temporal dependencies among the previous perceptions are exploited.

Action

The set of actions that the agent can execute is denoted A. Actions, $a_i \in A$, are used for changing the world state. Actions can be identified as being of either manipulatory, navigational, or sensing type. Accordingly, three action classes are introduced. Actions from the manipulatory action class change the state of the manipulator, which in some cases also changes the state of the environment. Actions from the navigational action class correspond to the movement of the agent to a new location. Sensing actions correspond to the acquisition of information about the world state.

Reasoning

Given perceptions of the world state, the agent performs reasoning, ρ, to determine what action sequence to execute.

Effectuating

The robot is using effectors to translate actions into changes of the world state. Effectuating, ε, therefore maps actions to world state changes.

Functional description of an agent

In a one-agent application, it is possible to describe perceptions as functions of actions or vice versa. This is illustrated in *Fig.2*.

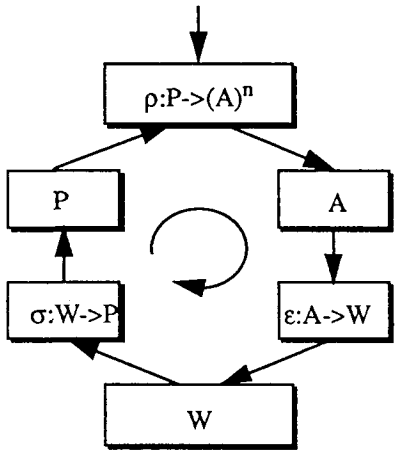


Fig. 2. Functional description of an agent

Following is the interpretation of *Fig.2*: Perceptions, P, are mapped by reasoning, ρ, to an appropriate action sequence $(A)^n$. The effectuating, ε, maps each action to a new world state, W. Sensing, σ, maps the world state to perceptions, P_i.

Let C_k denote k:th reasoning-effectuating-sensing cycle, that is

$$C_k \equiv \sigma_k\left(\varepsilon_{k,n_k}\cdots\varepsilon_{k,2}\varepsilon_{k,1}\right)\rho_k = \sigma_k\varepsilon^{n_k}\rho_k \qquad (3)$$

(3) has the following interpretation:

By reasoning, ρ_k, the robot decides to execute an action sequence consisting of n_k actions, which are send to the effectuating, ε. Thereafter the robot uses sensing, σ_k, to acquire information about the resulting world state. Accordingly, the k:th perception can be written as:

$$P_k = C_k(P_{k-1}) = C_kC_{k-1}\cdots C_1(P_0) \qquad (4)$$

In applications where the agent corresponds to an autonomous robot it is possible to describe the mapping (ρ) explicitly by a mathematical function. Accordingly, it is possible to, given a perception vector P_i and the mapping ρ, determine what action sequence will be executed consequently. This is possible since machine reasoning consists of well-defined deterministic operations.

For applications where the agent corresponds to a teleoperated robot (or where man/machine cooperative decision making[15] is used), the value of the mapping, ρ, corresponds to the action sequence (partly) decided upon by the human expert. Since it is hardly possible to establish a deterministic model of the reasoning process involved, the true mapping function, ρ, is (partly) unknown for teleoperated robots.

However, in either case, the impact of an action on the world state will be the same. This implies that the perceptions dependency on actions is invariant under different reasoning approaches. Consequently, this suggests that the same principles for maintaining the information quality can be utilized for both agent types.

4 Information quality

Having introduced a set of perceptual classes, a measure of the information quality of each perception is needed. The measure should reflect both the accuracy and relevance of a perception. External perceptions are commonly maintained at three abstraction levels to reduce complexity and enable inference. The lowest level contains numerical, the second geometric, and the third symbolic information respectively[3]. Consequently, the information quality measures at two distinct abstraction levels will differ. In this paper, we will consider only the two lowest abstraction levels. Internal perceptions, describing the state of the robot, are often of limited complexity. Hence, only a numerical model is needed. For example, if ignoring dynamical properties, the state of a mobile robot equipped with a manipulator arm can be described by a 9-dim. vector (3 dim. for describing the robot pose and 6 dim. for describing the position and orientation of the end effector).

Accuracy

Perceptual accuracy is a static attribute that is determined when the perception is created. It is static because no future event can affect the accuracy of an already made measurement. If perceptions are treated as vectors, one common way to represent accuracy for numerical perceptions are by the corresponding covariance matrices. In this way, Kalman filtering techniques can be utilized to gener-

ate accurate state estimates. A classical paper on this is the one by Smith, Self, and Cheeseman[23] which provides a framework for handling robot positional uncertainty, and may be extended to also handle robot arm positional uncertainty.

Relevance

Besides being more or less accurate, a perception will also be more or less relevant. For example, a temperature measurement from one part of a building tells very little about the temperatures in other parts of the building, although the temperature was measured with high accuracy. Furthermore, after some time that particular temperature measurement tells very little about the current temperature even in the position where it was obtained. These two facts correspond to the limited spatial and temporal applicability range of perceptions. Thirdly, assume that the agent that did the previous temperature measurement decides to open a window. Provided it is a temperature difference between the inside and outside of the building, this action will reduce the relevance of the previous measurement. Thus, manipulatory actions executed by the agent is another cause of variation in relevance.

While the first and third example illustrate the dependency of the relevance on the executed navigational or manipulatory actions of the agent respectively, the second example illustrates that the application is not an ideal one-agent application. Since no real-world application is an ideal one-agent application, a perception aging function, monotonously decreasing with time, must be used.

The presented examples describe relevance for a numerical perception. In *Section 7* a relevance measure for geometric perceptions, based on the maximum entropy principle, will be described.

Information quality measure

Our approach assumes a bidirectional dependency between perceptions and actions. This is an extension to the approach by Erdmann[8], who assumes a unidirectional perception/action dependency.

Section 3 introduced a description of the k:th perception as the result after a reasoning-effectuating-sensing sequence has been executed after an initial perception P_0 has been created.

The information quality of a perception vector P_k is denoted $QI(P_k)$. The evaluation of the information quality is as

$$QI(P_k) = (q_1(p_1), q_2(p_2), \ldots, q_r(p_r)) \quad (5)$$

where $q_i(p_i)$ is evaluated as

$$q_i(p_i) = max\ \{q_i(p_{i,k_x})\} \quad (6)$$

where k_x is iterated over all creation times of the perceptions in the i:th perceptual class.

Using (3) and (4), (5) can be rewritten as

$$QI(P_k) = QI\left(\sigma_k \varepsilon^{n_k} \rho_k P_{k-1}\right) \quad (7)$$

The index n_k must be selected as to satisfy the condition

$$\left(QI\left(\sigma_k \varepsilon^{n_k} \rho_k P_{k-1}\right) > T\right) \wedge QI\left(\sigma_k \varepsilon^{n_k+1} \rho_k P_{k-1}\right) < T \quad (8)$$

is satisfied, where T is the information quality threshold value. This condition means that executing n_{j+1} actions in a sequence will result in an information quality value below the threshold value. Since robot malfunction corresponds to the failure to execute a particular action, the threshold value must take into account that for some actions an action failure is harmless while for irreversible hazardous actions a successful action execution is essential. Thus, to permit the execution of an action, the information quality of the perceptions must exceed its corresponding threshold value, T.

Predictions

Knowledge about the statistical properties of the effectors enables the prediction of the environmental state resulting from an executed action. Furthermore, with information about the statistical properties of the sensors, it is possible to predict how the resulting environmental state should be perceived if a new perception where obtained.

Given a perception and an action to execute, it is possible to predict what should be perceived after the action has been executed. The uncertainty in this prediction is determined by the statistical properties of the stochastic mapping. If this uncertainty is considered to be low enough, an additional action may be executed, with a corresponding new prediction. The uncertainty in this new prediction is higher than for the previous prediction. This can be iterated as long as the uncertainty in the prediction is low enough. When, at last, the prediction will be too uncertain, a new perception must be generated.

The agent can predict the true world state resulting after each executed action by using knowledge about the most

probable outcome of the action. The resulting prediction of the world state after n_a actions (after time k) have been executed is denoted $\hat{P}_{k,a}$. Thus,

$$\hat{P}_{k,a} = \varepsilon^{n_a} \rho_k P_{k-1} \tag{9}$$

The information quality of the prediction is denoted $QI(\hat{P}_{k,a})$, that is

$$QI(\hat{P}_{k,a}) = QI\left(\varepsilon^{n_a} \rho_k P_{k-1}\right) \tag{10}$$

where n_a must satisfy the condition

$$\left(QI\left(\varepsilon^{n_a} \rho_k P_{k-1}\right) > T\right) \wedge \left(QI\left(\varepsilon^{n_a+1} \rho_k P_{k-1}\right) < T\right) \tag{11}$$

The evaluation of (11) is done in a way similar to (6).

5 Sensor planing

Both autonomous and tele-operated robots require a criterion for when to acquire new information. According to the previous discussion this is necessary when the quality of the information is so low that the reasoner is likely to draw the wrong conclusions about the situation. This corresponds to some perception being too inaccurate or too irrelevant. A sensor planning algorithm should keep track of the resulting accuracy and relevance of the perceptions as actions are executed, and enforce a sensing action if the accuracy or relevance has become too low.

In sensor planning, an important difference between autonomous and tele-operated robots is their type of reasoning (ρ). For autonomous robots, a well-defined mapping from perceptions to actions is used, while for tele-operated robots, a human expert decides upon what action to execute next. In effect, autonomous robots may use (8), where the resulting information qualities are calculated before the sequence is executed. For tele-operated robots, the situation is different. Here, the sensor planning algorithm has no knowledge about what action will be executed next. Therefore, it must keep track of the information quality values during operation and stop the robot if the information quality has become too low. This implies that (11) should be used instead.

6 Maximum entropy methods

The information theoretical entropy concept has been applied in a variety of scientific disciplines. For a survey of entropy optimization techniques, we refer to Kapur and Kesavan[14]. In robotics, Saridis[20, 21, 22] and Valavanis[24] have described robot systems that use the concept of entropy as a global performance measure. In their approach, optimal robot behavior is achieved through the minimization of the total system entropy. Sanderson[19] uses the entropy concept to describe the complexity of differently shaped geometrical objects, measured in bits. Finally, the thorough study of relations between information theory and search theory conducted by Pierce[18] has inspired the development of the relevance measure for the case study described in *Section 7*.

7 Case Study - Sensor planning on a tele-operated indoor intervention robot

This case study demonstrates how the previously introduced concepts can be instantiated in a real-world application for a mobile robot equipped with a manipulator arm. The robot obtains information about the external state by using laser and video cameras. Information about the internal state is obtained through odometry and angle counters on the joints on the manipulator arm. This robot type is very general, but by constraining either the mobility or the manipulability capabilities, more restricted robot types are obtained. In the case study, a representative set of perception and action classes are introduced. Furthermore, to properly control the system, the robot must have a system model with low discrepancy. Because of limited computational power, the model must have as low complexity as possible. In order to establish a compact but yet useful model, it is important to exploit structure in the robot operations. In the case study, a composite 2D/3D model developed for the discussed robot type is elaborated.

Robot operation

During operation, the state changing actions executed by the robot can be classified as being either navigational or manipulatory. Navigation corresponds to the movement of the robot to a new location, while manipulation corresponds to the reorientation of the manipulator arm (the purpose of the manipulator arm is to change the state of the environment). This suggests the introduction of the two action classes A_N and A_M, denoting the navigational and manipulatory action class respectively. When the robot executes a sequence of navigational actions, it is said to be in navigational mode. Similarly, when the robot executes a sequence of manipulatory actions, it is said to be in manipulatory mode.

Assuming a one-agent application, the world consists of two entities, agent and environment respectively. The state of the agent will be denoted the internal state while the state of the environment will be referred to as the external

state. Accordingly, measurements of the internal state will be referred to as gauging, reflecting the contact-type of internal measurements, while measurements of the external state will be referred to as sensing.

This gauging/sensing action class division is natural. First, the division conforms to the two-mode robot operation since only the internal state changes when in navigational mode. Second, the complexity of the internal state description is much lower than the complexity of the external state description. Consequently, a division is necessary of the external state description into representations at different abstraction levels. As a comparison, it is possible to establish a basic description of the internal state using a 9-dim. vector, while sensor data of the external state may well contain 10,000 data points.

The introduced four action classes is listed in (12):

$$\{A_S, A_G, A_N, A_M\} \tag{12}$$

with the following respective interpretation:

- A_S corresponds to Sensing actions, which measure the external state.
- A_G corresponds to Gauging actions, which measure the internal state.
- A_N corresponds to Navigational actions, which change the global state.
- A_M corresponds to Manipulatory actions, which change the local state.

Perceptual classes

Having introduced action classes in the previous section, this section presents a step-wise partitioning of the world description into a set of perceptual classes.

Perceptions describe different aspects of the world and thus represents a world model. Typically, the data from one or more sensor is refined and transformed into a perception (a perception is similar to *virtual sensor*[4] and *logic sensor*[15] respectively). In turn, a perception may be used to generate a more abstract perception, or be combined with previous perceptions to generate a perception with higher accuracy.

In the previous section, the two information acquiring action classes A_G and A_S was introduced. The corresponding perceptions resulting from information acquiring actions from respective action class are denoted internal and external perceptions respectively. As mentioned, the mobile robot can be viewed as being in either a navigational or a manipulatory operation mode. This observation suggests the partitioning of the perceptions into global vs. local perceptions. Global perceptions are perceptions that provide vital information when in navigational mode, while local perceptions are perceptions that provide vital information when in manipulatory mode. This leads to the division of the perceptions into four perceptual classes (Table 7-1).

	Local	Global
Internal	$P_{I,L}$	$P_{I,G}$
External	$P_{E,L}$	$P_{E,G}$

TABLE 7-1 Perceptual Classes

For the robot at hand, examples of perceptions belonging to each perceptual class are suggested below:

- $P_{I,L}$: The pose of the robot arm
- $P_{I,G}$: The pose of the robot
- $P_{E,L}$: The pose of a manipulable object
- $P_{E,G}$: The spatial description of the building

2D representations of the global perceptions P_{EG} and P_{IG} suffice in most applications since the robot moves on almost flat surfaces and detected obstacles may be assumed to have infinite height. Having infinite height implies that it suffice to describe their projections on the 2D plane. Thus, navigational actions are described within a 2D model, where the pose description contains three parameters (two for position and one for orientation).

General manipulatory actions involve manipulation of objects arbitrarily oriented in 3D space. Since the range of the robot arm is limited, a 3D model will be reasonable. For each object that is to be manipulated, a 3D description of its closest environment is used. The local 3D descriptions are connected to the global (navigational) 2D model, thus providing a composite 2D/3D model.

Accuracy

Smith, Self, and Cheeseman's formalism[23] may be applied for estimating the internal state, consisting of the robot pose (x, y, φ), where x, y describe the position and φ the orientation of the robot, and of the position and orientation of the robot arm $(x, y, z, \phi, \theta, \psi)$, where x, y, z corresponds to the position and ϕ, θ, ψ corresponds to the orientation of the end effector using euler angels. The introduced information quality threshold value, T, will in

this case correspond to a matrix where the elements should not be exceeded by the corresponding elements in the covariance matrix for the 9-dim. vector describing the compound internal state.

Relevance

The relevance of a perception is affected by spatial and temporal factors. While the former concerns the topology of the environment, the latter provides means for compensating for the one-agent assumption in applications actually containing several agents. Typically, the aging of the perceptions is modelled by simply scaling the perceptions with a monotonously decreasing weight function. The spatial factors affecting the relevance of perceptions of the global external state will be elaborated below, yielding a relevance measure applicable to laser range data*, although it may be modified to be used for sonar range data instead.

An assumption being made is that the gathering of spatial information is costly in time and resources. Therefore, new spatial information should be acquired only when absolutely necessary. To determine when this is the case, a measure indicating the relevance of a perception is needed.

Consider the situation in *Fig.3*. Spatial information obtained from position p1 will lack information about the region N.E. of p2. The sector indicated in the figure will accordingly be referred to as a *hidden sector*. The hidden sector indicates that the spatial information from p1 is not as relevant for describing the environment of p2 as it is for describing the environment of p1. If the only spatial information available is the one from p1, it is impossible to tell whether the spatial information not contained in p1 is important or not. Let R denote the statement "Relevant information" and $\neg$R the statement "Not relevant information" (e.g, in search operations, indications of the object being searched for are considered relevant). Letting $p_i(R)$ denote the probability that scan direction i contains relevant information, then Laplace's principle of insufficient reason states that $p_i(R) = p_i(\neg R) = 1/2$ if no prior information is available.

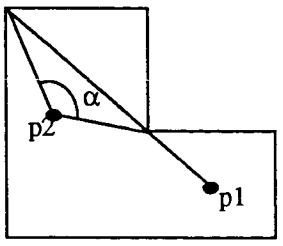


Fig. 3. Relevance of spatial information, hidden sector

Assume that the laser measures range in M consecutive directions in a horizontal plane. Then M probability functions are introduced. It is assumed that after a measurement has been made, it is clear whether a particular direction contains relevant information or not. Thus, after a measurement $p_i(R) \in \{0, 1\}$. For large M, the angle resolution will be high enough to justify the assumption that no new information will come out of an additional measurement of a previously scanned region.

In the case of *Fig.3* this implies that the only new information coming out of a measurement from position p2 is information about the region that is not described by the measurement from p1. Assuming the angle between consecutive scan directions to be $\Delta\varphi$ and that the hidden sector has angle α, a scan from p2 will provide new information from $d = \alpha/\Delta\varphi$ directions. Letting $\bar{p}$ and $\bar{q}$ denote the ensemble of probability functions corresponding to the measurements from p2 and p1 respectively, then the amount of new information obtainable from p2 given the information from p1 is

$$H(\bar{p}|\bar{q}) = -\sum_{i=1}^{d}\sum_{j} p_i(j)\log(p_i(j)) \qquad (13)$$

where $j \in \{R, \neg R\}$. Using the maximum entropy principle, the sum in (13) is equal to $d \cdot \log 2$, which is the amount of new information obtained if scanning from position p2, or equivalently stated the amount of information about p2:s environment not included in the spatial description from p1. As is seen, the obtained information is proportional to the width of the hidden sector.

The amount of information obtained from the initial scan is $M \cdot \log 2$. No later scan will give that much information. This suggests that the information quality thresholds for the external perceptions should belong to the interval $[0, M \cdot \log 2]$.

* Using range data from either laser or sonar is a common way to build up maps of indoor environments[3, 5, 9]

In the general case there are k hidden sectors, corresponding to k sub-intervals in the interval 1,2,..., M that provide new information (see *Fig.4*).

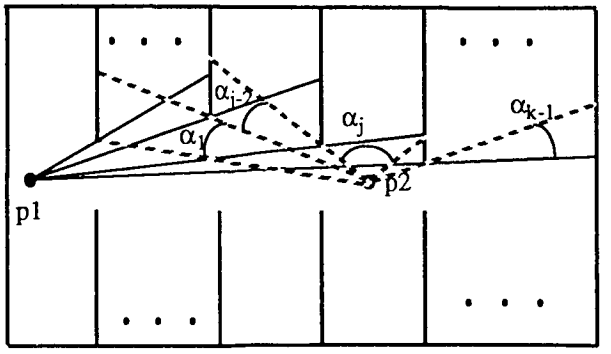


Fig. 4. k hidden sectors

Extending (13) to cover k hidden sectors yields

$$H(\bar{p}|\bar{q}|) = -\sum_{h=1}^{k}\sum_{i=1}^{d_h}\sum_{j} p_i(j)\log(p_i(j)) \qquad (14)$$

which, according to the maximum entropy assumption is equal to

$$\log 2 \cdot \sum_{h=1}^{k} d_h \qquad (15)$$

Notice that if the laser range data is segmented by using polyline segmentation[6], the hidden sectors can be calculated directly, since the information needed to calculate the hidden sectors is information about the coordinates of the endpoints of the line segments.

Information quality

As mentioned, there is a close connection between perceptions and actions in one-agent applications. This section describes this dependency. *Fig.5* captures the dependency between the action and perception classes. The actions positive/negative influence on the information qualities of the perceptions are indicated by the "+" or "-" superscript.

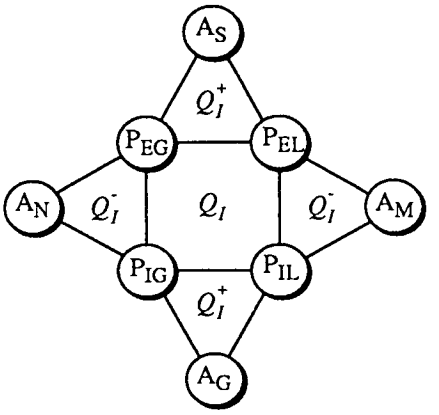


Fig. 5. Perception/Action class dependency

In *Fig.5*, the dependency within a pair of perception/action classes is bidirectional. The information qualities of the perceptions are either increased or decreased depending on what actions that have been executed. A similar dependency is present in the opposite direction since the information qualities of the perceptions must be sufficiently high in order for the reasoner to perceive the situation correctly and select proper actions. In *Fig.5*, the perceptual classes have the following interpretations:

- Internal global state (P_{IG}) corresponds to the position and orientation of the robot, and is measured by odometry.
- Internal local state (P_{IL}) corresponds to the position and orientation of the end effector (robot hand), and is measured by combining measures from angle counters on the joints of the robot arm.
- External global state (P_{EG}) corresponds to a line segment description of the robot environment. The description is created by line segmentation[5] of laser data, and is measured by a laser range finder.
- External local state (P_{EL}) is measured by vision (TV camera).

For navigation tasks, the operator uses the line segment description of the robot environment, while for manipulatory tasks views from TV cameras are used. The perception/action dependency may then be illustrated by the dependency graph in *Fig.6*, which emphasizes the two-mode operation. The nodes in the leftmost column correspond to sensors, the nodes in the middle column correspond to perceptions, and the nodes in the rightmost column correspond to actions.

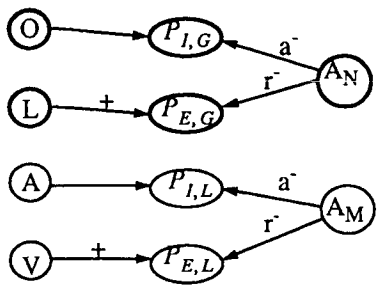


Fig. 6. Information quality dependency graph

Odometry (O) gauges the internal global state, the description of which is stored in $P_{I,G}$. New measurements cannot reduce the increasing positional uncertainty as the robot moves, although Kalman filtering techniques can reduce this problem. For this reason, the arrow from O to $P_{I,G}$ in *Fig.6* is unlabeled. Laser (L) senses the external global state, the description of which is stored in $P_{E,G}$. Acquiring new laser data improves the relevance of $P_{E,G}$, indicated by a "+"-sign on the arrow between L and $P_{E,G}$ in *Fig.6*. Angle counters (A) gauge the internal local state, the description of which is stored in $P_{I,L}$. Although new measurements cannot reduce the uncertainty in $P_{I,L}$, the uncertainty will not increase monotonously as the uncertainty corresponds to lashes in the joints. Vision (V) senses the external local state, the description of which is stored in $P_{E,L}$.

In this application, the external local states corresponds to local descriptions of particularly interesting regions in the environment (e.g. regions containing manipulatable objects). The relevance measure introduced in *Section 4* is intended for external local perceptions (which are 2D). A similar measure for the external local perceptions (which are 3D) may be developed based on the concept of hidden cones although this is not described in this paper.

Navigational actions, A_N, corresponds to the transport of the robot to a new position in the environment. The accuracy of the internal global perception $P_{I,G}$ is decreased due to accumulation of positional uncertainty. Also, the relevance of the external global perception is decreased, in accordance with the fact that the applicability of the external global perception may decrease as the robot moves away from the position at which the external global perception was generated.

Manipulatory actions, A_M, corresponds to the movement of the manipulator arm, which may affect the accuracy of the internal local perception and/or the relevance of the external local perception.

Finally, as mentioned, the increased uncertainty in the estimation in the robot position is impossible to eliminate by solely using odometry. However, it is possible to combine sensing and gauging actions to improve the information quality of $P_{I,G}$.[7].

Platform evaluation

Having a method for maintaining the information quality at an acceptable level, the problem arises what threshold values to use. By solving a task, typical for the application at hand, with distinct information quality threshold value settings, the setting providing the best trade-off between safety and speed should be used. To compare different robot configurations (i.e., using different sensors and effectors), the optimal parameter setting for each robot is determined whereafter their optimal performances are compared. If an information quality value of a perception could be held close to zero without affecting the performance, this suggests that the corresponding perception is of little use in the application and might be omitted. By assigning each action (a) a cost C(a), for example corresponding to execution time, the cost to use a particular platform is easily obtained by summing the cost of all actions in the action sequence that was executed during the mission. This value could then be combined with other factors, such as cost of sensors, number of errors during the mission etc., in a platform evaluation. Below, two platforms are described, one open-loop (without sensors) and one closed-loop (with sensors).

Open-loop robot platform

The mission cost (C_M) for this system is expressed as

$$C_M = \sum_{z \in \{N,M\}} \sum_i C\left(a_z^i\right) \tag{16}$$

As is seen, no sensing actions are executed, which corresponds to setting the information quality thresholds to zero. This system is appropriate only in a highly structured world.

Closed-loop robot platform

The mission cost in this case is expressed as

$$C_M = \sum_{z \in \{S,G,N,M\}} \sum_i C\left(a_z^i\right) \tag{17}$$

By repeating the solution of a task with different information quality threshold values, a compromise between safety and speed can be found. If a low information quality

threshold value provides fairly high safety, this suggests that the corresponding perception is of little use and may be omitted.

References

1 R.A. Brooks, A Robust Layered Control System For A Mobile Robot, IEEE J. Robotics Automation, RA-2 (1) (1986) 14-23.

2 R.A. Brooks, Elephants Don't Play Chess, Robotics Autonomous Systems 6 (1990) 3-15.

3 R. Chatila and J-P. Laumond, Position Referencing and Consistent World Modeling for Mobile Robots, Proc. IEEE Int. Conf. Robotics Automat., St. Louis, US (1985) 138-145.

4 B.R. Donald, J. Jennings, and R. Brown, Constructive recognizability for task-directed robot programming, Robotics Autonomous Systems 9 (1992) 41-74.

5 A. Ekman and D. Strömberg, Incremental map-making based on range images, M. Sc. Thesis, Defence Res. Establ., Sweden, 1992.

6 A. Ekman, D. Strömberg, and A. Lauberts, PSA - a Recursive Polyline Segmentation algorithm using an Eigenvector fit criterion, Technical Rep., Defence Res. Establ., Sweden 1993.

7 A. Ekman, D. Strömberg, and P. Klöör, Incremental map-making in indoor environments, Proc. IFAC workshop on Intelligent Autonomous Vehicles, Southampton, U K (1993).

8 M. Erdmann, Towards Task-Level Planning: Action-Based Sensor Design, Tech. Report, Carnegie Mellon University, Pittsburgh, PA, 1992.

9 O.D. Faugeras, Object representation, identification, and positioning from range data, Proc. 1st Int. Symp. Robotics Res., Cambridge, MA, (1984) 425-446.

10 B.R. Fox and K.G. Kempf, Planning, Scheduling, and Uncertainty in the Sequence of Future Events, in: J.F. Lemmer and L.N. Kanal, eds., Uncertainty in AI 2 (North-Holland, Amsterdam, 1988) 395-401.

11 I.J. Good, Dynamic Probability, Computer Chess, and the Measurement of Knowledge, in: E.W. Elcock and D. Michie eds., Machine Intelligence 8 (John Wiley & Sons, New York, 1977) 139-150.

12 I.J. Good, Rationality, Evidence, and Induction in Scientific Inference, in: E.W. Elcock and D. Michie eds., Machine Intelligence 8 (John Wiley & Sons, New York, 1977) 171-174.

13 E.T. Jaynes, Where do we stand on maximum entropy, in R.D. Levine and M. Tribus, eds., The maximum entropy formalism (MIT Press, 1979) 15-118.

14 J.N. Kapur and H.K. Kesavan, Entropy Optimization Principles with Applications, (Academic Press Inc., 1992).

15 S. Lee, Intelligent Sensing and Control for Advanced Teleoperation, IEEE Control Systems, 13 (3) (1993) 19-28.

16 J.S.B. Mitchell, D.W. Payton, and D.M. Keirsey, Planning and Reasoning for Autonomous Vehicle Control, Int. J. Intell. Systems 2 (2) (1987) 129-198.

17 J. Pearl, Fusion, Propagation, and Structuring in Belief Networks, Artificial Intelligence, 29 (1986) 241-288.

18 J.G. Pierce, A new look at the relation between information theory and search theory, in: R.D. Levine and M. Tribus, eds., The maximum entropy formalism (The MIT Press, Cambridge, Ma, 1978) 339-402.

19 A.C. Sanderson, Parts Entropy Methods for Robotic Assembly System Design, in: P.M. Jackson ed., 1983 annual report, Robotics Institute, Carnegie Mellon University (Pittsburgh, PA, 1983) 33-40.

20 G.N. Saridis, Toward the Realization of Intelligent Controls, Proc. IEEE, 67 (8) (1979).

21 G.N. Saridis, Entropy Formulation of Optimal and Adaptive Control, IEEE trans. Automatic Control, 33 (8) (1988) 713-721.

22 G.N. Saridis, Analytic Formulation of the Principle of Increasing Precision with Decreasing Intelligence for Intelligent Machines, Automatica 25 (3) (1989) 461-467.

23 R. Smith, M. Self, and P. Cheeseman, Estimating Uncertain Spatial Relationships in Robotics, in: J.F. Lemmer and L.N. Kanal eds., Uncertainty in Artificial Intelligence 2 (North-Holland, Amsterdam, 1988) 435-461.

24 K.P. Valavanis and G.N. Saridis, Intelligent Robotic Systems: Theory, Design and Applications, (Kluwer Academic Publishers, 1992).

25 L.A. Zadeh, Fuzzy sets as a basis for a theory of possibility, Fuzzy Sets and Systems 1 (1978) 3-28.

The Network Data Delivery Service: A Real-Time Data Connectivity System

Gerardo Pardo-Castellote

Aerospace Robotics Laboratory
Stanford University
Stanford, California 94305

Stan Schneider

Real-Time Innovations, Inc.
954 Aster
Sunnyvale, California 94086

Abstract

The Network Data Delivery Service (NDDS) is a novel network data-sharing system. The NDDS system builds on the model of information producers (sources) and consumers (sinks). Producers register a set of data instances that they will produce, unaware of prospective consumers and "produce" the data at their own discretion. Consumers "subscribe" to updates of any data instances they require without concern for who is producing them. The routing protocol is connectionless and nearly "stateless"; all data producer and consumer information is dynamically maintained. Thus network reconfigurations, node failures, etc. are handled naturally.

This scheme is particularly effective for systems (such as distributed control systems) where information is of a repetitive nature. NDDS features the ability to handle multiple producers, consumer update guarantees, notifications vs. polling for updates, dynamic binding of producers and consumers, user-defined data types, and more. NDDS is integrated into the ControlShell real-time framework, and is being used in several robotics applications as an effective means of information sharing between sensor systems, robot controllers, planners, graphical user interfaces, simulators etc.

This paper describes the philosophies behind the NDDS system, and details an example application of a dual-arm robotic system capable of planning and executing complex actions under control of an interactive user interface.

1 Introduction

Many control systems are naturally distributed. This is due to the fact that often they are composed of several physically distributed modules: sensors, command, control and monitoring modules. In order to achieve a common task, these modules need to share timely information. Robotic systems are a prime example of such distributed control systems.

These information sharing needs are common to many other application environments such as databases, distributed computing, parallel computing, transaction systems, etc. However, distributed control applications have some unique requirements and characteristics:

- Data transactions in control applications are often time-critical. To be useful for control purposes data must get from its source to its destination with minimum delay.
- There is often the need to synchronize computation to the arrival of new data. For example the control command may need to be updated only when new sensor data arrives. A collision-avoidance plan may need to be re-evaluated when a new obstacle is detected, etc.
- A significant portion of the data flow is repetitive in nature. This is true of sensor readings, motor commands etc. For this type of data, data loss is often not critical: sending data is an idempotent operation and new updates just replace old values. This property suggests that considerable overhead can be avoided by using a data transfer paradigm that exploits these facts.
- There are often multiple sources of (what may be considered) the same data item. For example, a robot command might be generated by a planner module as well as a tele-operation module. In the same way there can be many data consumers. A robot and a simulator are both sinks of "command-data." The network of data producers and consumers may not be known in advance and may change dynamically.
- Data requirements are ubiquitous and unpredictable. It is often very difficult to know what data will be required by other modules. For instance, force-level measurements—normally used only by a low-level

Copyright c 1993 by Real-Time Innovations, Inc. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

controller—may be required by a sophisticated high-level task planner in the future. The architecture should support these types of data flow. Thus, vital data should be accessible throughout the system.

- Most data flow can and should be anonymous. Producers of the sensor readings can usually be unaware of who is reading them. Consumers may not care where the data they use came from. Since it is not essential, hiding this information increases modularity by allowing the data sources and sinks to change transparently.

These requirements are sufficiently unique to deserve special treatment. To address all this needs we have developed the *Network Data Delivery Service* (NDDS).

NDDS provides transparent network connectivity and data ubiquity to a set of processes possibly running in different machines. NDDS allows distributed processes to share data and event information without concern for the actual physical location and architecture of their peers[1]. NDDS allows its "clients" to share data in two ways: subscriptions and one-time queries.

NDDS supports "subscriptions" as a fundamental means of communication. In the application context described, subscriptions have some fundamental advantages over other information sharing models (such as client-server or shared-memory). Subscriptions cut in half the data latency over query/response type models and it allows synchronization on the latest available information as soon as it is produced.

NDDS supports multiple information sources (producers) and users (consumers). It provides clear semantics for multiple-producer conflict resolution, provides support for and guarantees multiple update rates (as specified by the consumers), and uses decaying state at all levels to ensure inherently robust communication.

NDDS is into the *ControlShell* real-time programming framework [4, 7] and is being been used in several applications including the control of a two-armed robotic system [6], an underwater vehicle [10], and a self contained, two-armed space robot originally described in [9].

2 Relation to Other Research

There is a large body of literature covering information sharing in distributed computer systems [1, 2]. More recently new schemes have been developed to address the specific needs of distributed control applications (see citations below).

[1] NDDS is being used to communicate between Sun, HP and DEC workstations as well as VME-based real-time processors running the VxWorks operating system.

Several issues are relevant when comparing different data-sharing approaches.

- Abstraction presented to the user. How natural and appropriate is it for the specific domain of distributed robotic systems?
- Robustness. Recovery from computer and communication failures.
- Flexibility and Expandability. How easy is it to add/replace component modules. Can it be done *dynamically*, while the system is in operation?

In the last few years, several distributed data-sharing schemes have been developed to address many of the issues raised in the introduction among these MBARI's Data Manager [5], CMU's TCX [3], Rice University's TelRIP [11] and Sparta's ARTSE [8] all offer network-transparent connectivity across different platforms and support *subscriptions* as means of communication where multiple consumers can get updates from a single producer. Of these, only the Data Manager, provides support for multiple (consumer-specified) update rates. And only TelRIP supports multiple producers of a single data item. None of the above architectures combine the above facilities with NDDS's fully-distributed, symmetrical implementation (absence of privileged nodes) nor use a restartable handshake-free stateless protocol.

3 The NDDS Communication Model

The NDDS system builds on the model of information producers (sources) and consumers (sinks).

Producers register a set of data instances that they will produce, unaware of prospective consumers and "produce" the data at their own discretion. Consumers "subscribe" to updates of any data instances they require without concern for who is producing them. In this sense the NDDS is a "subscription-based" model. The use of subscriptions drastically reduces the overhead required by a client-server architecture; Occasional subscription requests, at low bandwidth, replace numerous high-bandwidth client requests. Latency is also reduced, as the outgoing request message time is eliminated.

NDDS identifies data instances by a name (the *NDDS name*). The scope of this name extends to all the tasks sharing data through NDDS. Two instances with the same *NDDS name* are viewed by NDDS as different updates of the *same* data instance and are otherwise indistinguishable to the client. If two data instances must be distinguished by any NDDS client, they must be given a different *NDDS name*.

Function	Action
NddsRegisterProducer	Register and specify producer parameters
NddsProduce	Add an instance produced by the producer
NddsSampleProducer	Take a snap-shot of all the instances produced by the producer. For immediate producers also send updates to all consumers.
NddsRegisterConsumer	Register and specify consumer parameters
NddsSubscribeTo	Add a subscription to a consumer. Specify a call-back routine to be called on updates
NddsReceiveUpdates	Poll the consumer for updates. Will result on call-back routines being called when applicable. Required only of polled consumers.
NddsQueryInstance	Issue a one-time query.

Table 1: **Functional interface to produce, consume and query data.**

Producing data involves three phases: Registering (declaring) a producer, declaring the instances the producer will produce and sampling the producer. Receiving data updates also involves three phases: Registering (declaring) a consumer, declaring the instances that the consumer subscribes to along which the action to be taken and lastly receiving the updates. The last phase is only required for polled consumers.

NDDS requires all data instances to be of a known type. NDDS has some built in types (such as strings and arrays) but most data flow consists of user-defined types. Creating a new *NDDS type* involves binding a new type-name with the functions that will allow NDDS to manipulate instances of that type.

NDDS treats producers and consumers symmetrically. Each node maintains the information required to establish communications. Producers inform prospective consumers of the data they produce. Consumers use this information to either subscribe to data or issue one-time queries. Table 1 lists the steps involved in becoming a producer or consumer of data.

3.1 Producer Characteristics

A producer can be compared to a multi-channel Sample-and-Hold. It is associated with a set of object instances (similar to the signal channels) that get sampled synchronously. Sampling a producer takes a sample of the values of each data item the producer has associated with it. The data is either immediately distributed (for *immediate* producers) or saved for later distribution (*delayed* producers).

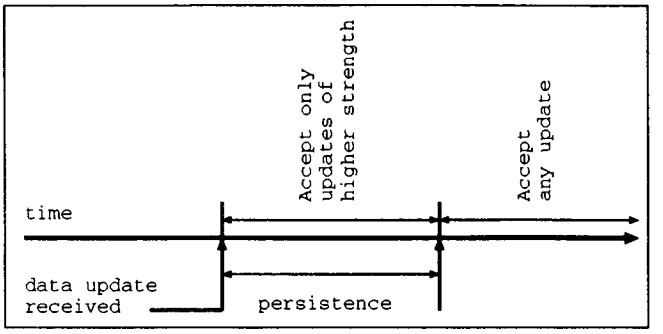


Figure 1: **Multiple producer conflict resolution.**

NDDS resolves the multiple-producer conflict by characterizing each producer with two properties: the producer's strength *and its* persistence. *When a data update is received for some object instance, it is accepted if either the producer's strength is greater (or equal) to that of the producer of the last update for that instance or, the time elapsed since the last update was received exceeds the persistence of the producer of the last update. In essence the strength is like a priority and the persistence is the duration for which the priority is valid.*

A producer is characterized by three parameters: its *production rate*, its *strength* and its *persistence*. The strength and persistence parameters are used to resolve multiple-producer conflicts. Their meaning is illustrated in Figure 1. A producer's data is used while it is the strongest source that hasn't exceeded its persistence. Typically a producer that will generate data updates every period of length T, will set its persistence to some time T_p where $T_p > T$. Thus, while that producer is functional, it will take precedence over any producers of less strength. Should the producer stop distributing its data (willingly or due to a failure), other producers will take over after T_p elapses. This mechanism establishes an inherently robust, *quasi-stateless* communications channel between the strongest producer of an instance and all the consumers of that instance.

3.2 Consumer Characteristics

Consumers are *notification based*. They subscribe to a set of instances (identified by their *NDDS name*) by providing *call-back functions* for each instance they sub-

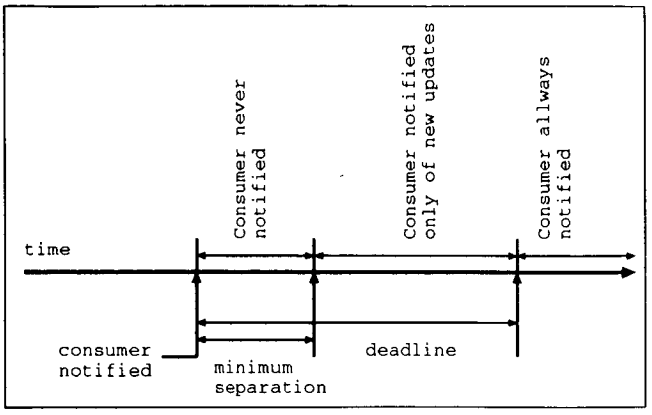


Figure 2: **Consumer notification rate.**

The NDDS characterizes consumers requests for periodic updates with two properties: the consumer's minimum separation *time and its* deadline. *Once the consumer is called with an update for an object instance, it is guaranteed not to be notified again of the same instance for at least the minimum separation time. The deadline is a the maximum time the consumer is willing to wait for a new update. Even if new updates haven't arrived, the call-back routine will be called when the deadline expires.*

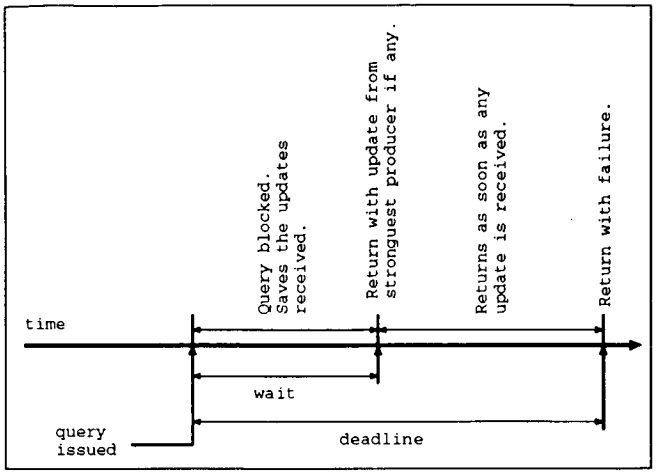


Figure 3: **One-Time Query Parameters.**

A one-time query specifies two parameters: the wait *time and the* deadline. *A query will block for at least wait time during which the data arriving from the producer with higher strength is saved. At the end of the wait time the query returns if any data has been received. Otherwise, query remains blocked until either an update comes or the deadline expires (whichever comes first).*

scribe to. When a data update for a subscription arrives, the call-back function of every consumer is called with the data as a parameter.

Two consumer models are supported: *immediate* and *polled*. An immediate consumer will be called back as soon as the data update arrives. A polled consumer will not be called back until it itself "polls" for updates. Consumers are characterized by two parameters, the minimum separation and the deadline (see figure 2). These parameters are used to regulate consumer update rates. Consumers are guaranteed updates no sooner than the minimum separation time and no later than the deadline. Typically the minimum separation protects the consumer against producers that are too fast whereas the deadline provides a guaranteed call-back time that can be used to take appropriate action (the expiration of the deadline typically indicates lack of producers or communications failure).

3.3 One-Time Queries

A client task may issue one-time queries for specific NDDS data instances. Queries are blocking calls. Aside from specifying the name and type of the NDDS data instance, a query contains two parameters: the *wait* and *deadline* illustrated in figure 3. These parameters regulate the tradeoff between returning as soon as data becomes available and waiting for "better" data. The use of these parameters make the latency of this call predictable, allowing its use from real-time application code. Typically the wait is set to be long enough to account for communication delays from all data producers to the consumer. The deadline provides a guaranteed call-back time in case no responses arrive. Setting a wait time to 0 causes the first response to be accepted.

4 Implementation

NDDS is symmetrically distributed, that is, there are no "special" or "privileged" nodes nor name servers. All NDDS nodes are functionally identical and each node maintains its own copy of the NDDS database and contains the helper processes necessary to implement the communication model described above.

NDDS uses UDP as a means of communication. Data is encoded using XDR to allow communications between computers with different data representations.

4.1 Architectural Overview

An NDDS *node* is composed of one or more NDDS client processes (each with its respective NDDS Server Daemon) a copy of the NDDS database and three daemon (helper) processes that maintain the database and implement the

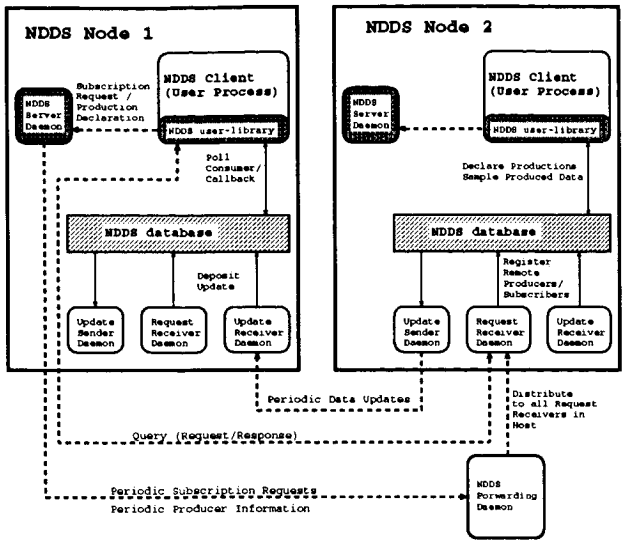


Figure 4: **Communication between NDDS nodes.**

A NDDS node *is composed of one or more NDDS* clients *and three helper daemons that share a local copy of the NDDS database. Each NDDS* client *has a private NDDS Server Daemon that informs other NDDS* nodes *of the productions and subscriptions of the client. The three helper daemons are responsible for maintaining the NDDS database, sending updates to remote subscribers, receiving updates and servicing queries. There is one NDDS Forwarding Daemon per Network host.*

NDDS communication model described above. See figure 4.

The user task becomes an NDDS client by linking to the NDDS library. Each NDDS client process spawns a private NDDS Server Daemon process that will assist in establishing the subscriptions and informing the peer nodes of the user productions. There is at most one NDDS node per address space so in operating systems that support shared memory threads (for example VxWorks), several NDDS client processes may belong to the same node (sharing the same copy of the NDDS database and helper daemons[2]).

The following is a functional description of the different daemons:

- **NDDS Forwarding Daemon** (NFD). There is one NFD per network host. All the Request Receiver Daemons running on the host register with the NFD. Production notifications and subscription requests received by the NFD daemon are immediately forwarded to all the Request Receiver Daemon(s) running on the host.

- **NDDS Server Daemon** (NSD). Each NDDS client (user-task) spawns its private NSD. The NSD is responsible for periodically informing the all the other NDDS nodes of both the subscription requests and the productions of the NDDS client.

- **Request Receiver Daemon** (RRD). There is one RRD per NDDS node. The RRD is responsible for maintaining the remote subscriptions and productions in the NDDS database. Stale productions and subscription requests are aged and eventually dropped by the RRD. This daemon is also responsible for replying to one-time *queries* from other NDDS nodes.

- **Update Sender Daemon** (USD). There is one USD per NDDS node. The USD is responsible for sending the updates of locally produced data items to the subscribers in other NDDS nodes. This daemon also ensures that the the timing parameters requested by the consumer are met.

- **Update Receiver Daemon** (URD). There is one URD per NDDS node. The URD is responsible for receiving updates for the local subscriptions of the nodes. The URD solves multiple-producer conflicts and, in the case of *immediate* consumers, executes the callback routine(s) installed for that data item. The URD also ensures that the timing parameters requested by the each consumer in the node are met.

4.2 Data Management Overview

The NDDS database is replicated and maintained on each NDDS *node* by three helper daemons (the Request Receiver Daemon, Update Sender Daemon and Update Receiver Daemon). The database stores and cross references producers and the data they produce, consumers and the data they consume, remote productions, subscriptions requested by both the NDDS clients in both the local and remote NDDS nodes, etc.

Consistency between databases across different NDDS nodes is not necessary and requires no special effort. Temporary inconsistencies between databases may result on subscription requests (or queries) not reaching all the producers of a given data item and, as a consequence, different nodes may get data from different producers. A similar situation may result from the data loss due to communication failure. At worst this will be a transient situation that arises only if there are multiple producers of the same data.

[2] In operating systems that don't support shared memory threads, such as Unix, the helper daemons are not independent tasks but rather are installed as signal handlers.

All information about remote NDDS nodes is aged and is eventually erased unless it is refreshed. The NDDS Server Daemon associated with each NDDS *client* is responsible for the periodic refreshment of the information that concerns that NDDS *client* . This mechanism is inherently robust to remote node failures, communication dropouts and network partitioning. Furthermore, it requires no special recovery mechanisms.

5 Applications

This section describes how a distributed robotic application exploits NDDS unique facilities to build a modular expandable system that integrates planning into a two-armed robotic workcell [6]. The system is composed of four major components: user interface, planner, the dual-arm robot control and sensor system, and an on-line simulator. The graphical user interface provides high-level user direction. The motion planner generates complete on-line plans to carry out these directives, specifying both single and dual-armed motion and manipulation. Combined with the robot control and real-time vision, the system is capable of performing object acquisition from a moving conveyor belt as well as reacting to environmental changes on-line.

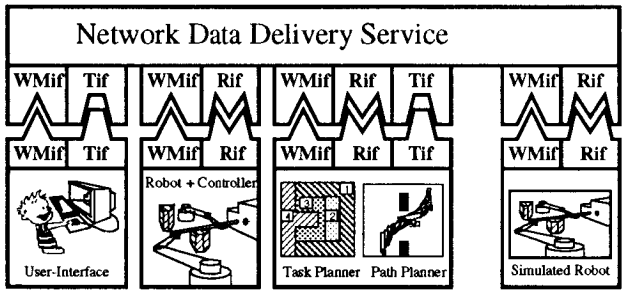


Figure 5: **Application Example: Task-Level control of a Two-Armed robot.**

This example shows the main application modules. Each module communicates using one or more of the three interfaces: The World Model interface (WMif), the Robot Interface (Rif) and the Task Interface (Tif). These modules are physically distributed. All the interfaces are built using the Network Data Delivery Service (NDDS). NDDS plays the role of a bus providing the necessary module interconnections.

The use of NDDS as the underlying communication mechanism provides unique benefits to this application without requiring any special programming.

- The different modules can be distributed across different computer systems (with different processor architectures and operating systems)[3].
- Several copies of the *same* module can be run concurrently. For example, the graphical user interface module can be run on several workstations. This allows multiple users to monitor the system and permits *simultaneous* interaction with the robot system.
- The graphical simulator module can *mascarade* as the real robot and allow independent testing of the remaining modules in the system. Any time the real robot goes on-line, its productions override those of the simulator[4] and all the remaining modules are now connected to the real robot.
- Should the planner be unable to generate adequate plans for specific situations due to limitations or malfunctions, a teleoperation module (under development) can override planner commands and take control of the robot.
- Future modules (such as the teleoperation-module mentioned above) can be *dynamically* added to the system even if it is already in operation or deployed[5].

6 Conclusions

This paper has presented NDDS, a unique data-sharing scheme that allows programs distributed on a computer network to share data and event information unaware of the location of their peers. These facilities provide fundamental new capabilities to distibuted control systems that use NDDS as the underlying communication mechanism. This paper has also discussed an application that uses NDDS to communicate between different modules that integrate planning into a two-armed robotic workcell. Several other applications are cited in the paper.

Acknowledgements

NDDS is being jointly developed by Stanford University and Real-Time Innovations, Inc. It is currently supported under ARPA's Domain-Specific Software Architectures (DSSA) program. The authors wish to expressly thank Dr. R. H. Cannon, Jr. for his guidance and leadership. The authors would also like to thank the many developers at Stanford and MBARI for their support and insight.

[3] This experiment has modules running on DEC Workstations, Sun Workstations and several VME-based real-time processors.

[4] Thanks to NDDS's multiple-producer conflict resolution mechanism.

[5] This facility may prove crucial for other current applications such as the undersea vehicles.

References

[1] Henri E. Bal, Jenniffer G. Steiner, and Andrew S. Tanenbaum. Programming languages for distributed computed systems. *ACM Computing Surveys*, 21(3):261–322, September 1989.

[2] Roger S. Chin et al. Distributed object-based programming systems. *ACM Computing Surveys*, 23(1):91–123, March 1991.

[3] Chris Fedor. TCX task communications. School of computer science / robotics istitute report, Carnegie Mellon University, 1993.

[4] Real-Time Innovations Inc. *ControlShell: Object Oriented Framework for Real-Time Software User's Manual.* 954 Aster, Sunnyvale, California 94086, 4.2 edition, August 1993.

[5] MBARI (Monterrey Bay Aquarium Research Institute. Data manager user's guide. Internal Documentation, 1991.

[6] Gerardo Pardo-Castellote, Tsai-Yen Li, Yoshihito Koga, Robert H. Cannon Jr., Jean-Claude Latombe, and Stan Schneider. Experimental integration of planning in a distributed control system. In *Preprints of the Third International Symposium on Experimantal Robotics*, Kyoto Japan, October 1993.

[7] S. A. Schneider, V. W. Chen, and G. Pardo-Castellote. Controlshell: A real-time software framework. In *Proceedings of the International Conference on Robotics and Automation*, San Diego, CA, May 1994. IEEE, IEEE Computer Society. (submitted).

[8] Sparta, Inc., 7926 Jones Branch Drive, McLean, VA 22102. *ARTSE product literature.*

[9] M. A. Ullman. *Experiments in Autonomous Navigation and Control of Multi-Manipulator Free-Flying Space Robots.* PhD thesis, Stanford University, Stanford, CA 94305, March 1993. Also published as SUDAAR 630.

[10] Howard H. Wang, Richard L. Marks, Stephen M. Rock, and Michael J. Lee. Task-based control architecture for an untethered, unmanned submersible. In *Proceedings of the 8th Annual Symposium of Unmanned Untethered Submersible Technology*, pages 137–147. Marine Systems Engineering Laboratory, Northeastern University, September 1993.

[11] J. D. Wise and Larry Ciscon. *TelRIP Distributed Applications Environment Operating Manual.* Rice University, Houston Texas, 1992. Technical Report 9103.

Robotics in a Controlled Environment Agriculture

Gaines E. Miles*
Purdue University
West Lafayette, Indiana

Jon D. Erickson†
National Aeronautics and Space Administration Lyndon B. Johnson Space Center
Houston, Texas 77058

ABSTRACT

Development and use of robotic systems are proposed to reduce the excessive human labor requirements associated with growing, processing, and recycling plants grown in controlled environments for agriculture and long duration human space missions. This paper presents a way of assessing requirements for intelligence-based robotics in an Advanced Life Support System (ALSS) involving both physicochemical and bioregenerative technologies.

In support of the application of system engineering principles to ALSS, leading to requirements assessment of robotic equipment, an approach is proposed which has four distinct components: (1) delineate the story of the mission in terms of processes with attendant types of resources needed, including options for use of robotic systems, (2) provide dynamic mathematical models of the biophysical processes involved, (3) provide 3-D animated graphical models of physical materials handling processes, and (4) provide systems dynamics models which, when used in simulation, reveal the implications for selected resource allocation schemes in terms of resources required to complete operational tasks. The simulations not only help establish ALSS design criteria such as production procedures and volume necessary for each crop, but they also may offer guidance to ALSS research efforts by identifying gaps in knowledge about procedures and/or biophysical processes.

* Professor of Agricultural Engineering

† Chief Scientist, Automation and Robotics Division, Member AIAA

Copyright © 1994 by the American Institute of Aeronautics and Astronautics, Inc. No copyright is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all right under the copyright claimed herein for Governmental purposes. All other rights are reserved by the copyright owner.

1. Introduction to Advanced Life Support Systems (ALSS)

NASA has studied controlled ecological life support systems (CELSS's) (Averner, 1990), or ALSS's, which are bioregenerative and based on a combination of biological and physicochemical processes since 1978. These systems may be used on future human space missions in low-Earth orbit, in transit to other planets, and on lunar and planetary surfaces. The purpose of an ALSS is to provide food and to replenish supplies of oxygen and water for long-term voyages in space. The basic physiological processes of plant growth (photosynthesis, nutrient uptake, translocation, transpiration, and respiration) combine carbon dioxide, minerals, and water to form food and inedible plant materials which must be recycled to supply nutrients for subsequent crops. Photosynthesis removes carbon dioxide exhaled by the astronauts and supplies oxygen for breathing. Transpiration adds odorless moisture to the air, where the water can be condensed and purified for consumption. Extensive work on the ALSS has been done by NASA Ames Research Center, Kennedy Space Center, Marshall Space Flight Center, and Johnson Space Center (Ming and Henninger, 1989).

The successful culture of higher-order plants requires the careful management of complex biological processes. An ALSS must have seeds (or other propagules), a source of mineral nutrients, water, a plant growth medium, and radiant energy (light). Labor and machinery are required to move materials from one process to another. It is necessary to monitor the state of the ALSS and diagnose and correct identified problems. Artificial intelligence offers supervised intelligent systems for monitoring and control of process operations and various powerful approaches to inspection, test, and diagnosis of problems, such as model-based diagnosis, as well as recovery from problem episodes.

Food production alone can be very labor intensive unless assisted by automated devices. During the 1920's, farmers without tractors, combines, and other machinery were able to produce enough food to feed themselves and one other person. Now, American farmers, fully mechanized with tractors, combines, and other machines, are capable of growing not only their own food, but enough for 50 others! When food processing, supply, and marketing efforts are included, about 1 in 5 Americans work in an agriculturally related business. Russian experiments conducted in BIOS 3 required an average of 4 hours per day per crew member to produce, process, recycle, and prepare food (Schwartzkopf, 1991). Biosphere 2 participants spent from 2 to 3 hours per day each operating their food production system.

In the ALSS, the processes are not restricted to production and processing, but must also include efforts to recycle inedible portions of plants into nutrients which can be utilized by succeeding crops. An ALSS for space exploration must be closed-loop. Nothing can be wasted nor allowed to accumulate in an unusable form. It is not sufficient to collect the garbage and bury it in landfills.

Biomass processing is an essential process which cannot be forgotten. It will consume labor, utilize resources, and have by-products which must be recycled to recover nutrients.

When one considers all of these activities required to keep the ALSS functioning, one will not be surprised when simulations indicate that robotic systems will be needed to avoid crew overload or even to complete the mission.

JSC is developing a Human Rated Test Facility (HRTF) (Henninger and Tri, 1993) in which to conduct year-long, high-fidelity tests of the ALSS, complete with four 90-day stays by crews of four. In order to gain the most from these experiments, a structured process of systems engineering (Petri, 1993) for the whole HRTF mission is being pursued within an integrated computer aided concurrent engineering environment (Erickson and Lawler, 1994), soon to be accessible via Internet. We want to design an HRTF that optimally considers the roles of humans, automation, and robotics. JSC will adapt and integrate, and develop where needed, automation and robotics technologies for applicability to the HRTF. JSC intends to carry out some of the robotic systems development through industry and university partnerships. As a key part of this systems engineering process, models and simulations are being developed. The models and simulations will be used to evaluate alternative processes, determine an optimum design of the HRTF, and to determine the most critical data to obtain during the human rated tests. The results of these experiments will be used to improve the models. Thus, the models and simulations will become a vehicle for structuring information and disseminating knowledge to peer scientists and engineers at other institutions.

The development and demonstration of a successful closed-loop ALSS is sure to have implications on how practically every community in America handles its "waste" and recyclables. Being able to disseminate this knowledge is just as important as discovering new information. Thus, models and simulations are key methodologies in ALSS research efforts.

2. Models and Simulations

Models are mathematical equations (including computer codes) which describe biological, physical, chemical, and operational processes. Lord Kelvin once said: "If you cannot model it, you don't understand it." The most useful models for design purposes are based on cause and effect relationships, not statistical correlations. Simulations are the process of utilizing the models to predict responses to changes in input conditions and/or equation parameters.

3. Biophysical Process Modeling

The biophysical processes in an ALSS not only have an impact on the design of the HRTF, but also influence the design of automation and robotics used as part of the HRTF. This is so for robotic systems because the crop production (volume of crop per unit area times area) directly impacts how much needs to be moved at what times. The rates of production determine the times for certain tasks.

The complexity of the biological, physical, and chemical processes in an ALSS requires mathematical (computer) models be developed for each candidate crop and each candidate process. At least as many as 10 different crops will be required to provide a balanced, nutritive diet for the astronauts. To meet operational constraints (such as carbon dioxide levels and peak energy demands), multiple stages of the same crop will be

likely. Photosynthesis, translocation of carbohydrates, transpiration, nutrient uptake and translocation, and respiration are a few of the processes which must be expressed as mathematical relationships between the environment (temperature, humidity, light intensity, etc.) and crop physiology (plant type, light interception, carbohydrate levels, etc.). By writing these expressions as cause and effect equations, the models can be used to accurately predict ALSS responses and to determine the optimum design. Because communities (or populations) of leaves, stems, roots, etc., are involved, the equations cannot be at a molecular detail, but must predict the overall population response. Just imagine the complexity of attempting to predict temperature (T), pressure (P), and volume (V) relationships in a gas by modeling molecular motions and integrating the impacts on vessel sidewalls. Since a much less detailed model (perfect gas law) has been proved to be valid, it would be a better choice to model the relationship between P, V and T. When developing models of systems, the correct level of detail (or hierarchical level) depends on the objectives or use of the model. The lowest level of detail for modeling will correspond to properties of leaves, stems, and roots in any given plant population. This tends to correspond to the objectives of using the model in the ALSS studies. (Of course, monitoring for the presence of various types of microbial activity takes us orders of magnitude higher in spatial granularity.)

At least two levels of modeling would be beneficial. At one level, the objective would be rather basic, fundamental questions concerning processes and rates of material transfer or conversion. Examples would include models which permit simulation of nutrient transfer from the transport/support media, which could be hydroponics or simulated lunar soil, or a manufactured soil mix based on degradable plant material. Simulation of the nutrient transfer rates would permit alternative designs of plant containers to be compared and evaluated. At the second level, the entire population of plants is considered, and questions such as percentage of light intercepted and rates of photosynthesis, respiration, leaf expansion, and translocation are answered on a canopy basis, not for individual plants.

4. Mission Simulations

One approach to analyzing an ALSS is to model all the tasks and resources required to meet prescribed performance specifications. Tasks include operations such as planting, transplanting, spacing of pots, gathering, threshing, cleaning, milling, baking, grinding, etc. Resources are labor, facilities, equipment, and supplies. For FY94, one objective is to build an initial mission simulation which can be upgraded when more information is available and which can be used in a short cycle run-break-fix approach to establish the feasibility and requirements of automation and robotics components of an ALSS. The software is based on a Mission Simulation and Analysis Tool (MSAT) (Erickson, Aucoin, and Dragg, 1992). These simulations will provide information on event timing, utilization of resources, and efficiency of rules which allocate scarce resources. The MSAT is unique, specialized software which permits close scrutiny of tasks proposed for a mission. Scenarios are encoded as parallel, interactive processes, and the outcome is reported as a sequence of daily reports of accomplishments and tasks not done for lack of critical resources. The simulator accurately accounts for the nonlinear effects of parallel, interactive processes, including contingencies and constraints, such as mission rules. MSAT keeps track of resource utilization and user-defined resource capabilities (equipment designs), and it won't allow simultaneous use of the same resource in different places. From the methodological point of view, MSAT is a model-based simulator which uses a representational framework with objects, processes, events, relations, and states of affairs as basic categories of information. This contrasts with other representational schemes which use fewer primitive types. A user has great flexibility in modeling mission processes; providing for options, including hazard conditions; and guiding the allocation of different types of resources to different types of processes (e.g., support, logistics, crop handling, maintenance, and so on).

In 1994, the MSAT simulation of the HRTF will primarily focus on the crop planting, growth, harvesting, and processing tasks performed by humans or those proposed for automation by robots. Times and options for performing all tasks will be estimated. Observation tables will be developed to describe day-by-day tasks that are to be accomplished during planting, growing, harvesting, processing and recycling of crops, and recycling of the growth medium. For each task, possible initial conditions will be established. For planting, initial conditions might be:

- Seed loose in a container,
- Seed tapes, or
- Seedlings in cell-flats.

Each step in the succeeding process will be identified and characteristics such as length of time, labor, growth area, and storage required for each step will be estimated during this early attempt to develop a model. Once the biophysical process models have been completed, their results will be dynamically linked to the operations knowledge base so that as estimates of resource requirements and ALSS productivity improve with additional research results, the design criteria for the mission simulators will be updated. Similar process descriptions are required for each candidate crop.

In addition to determining the amount of resources required for each mission scenario, simulations with these operations models will determine the amount of materials handling necessary to produce, harvest, and process edible and inedible biomass and recycle the plant growth medium for several alternative configurations of equipment and the HRTF layout. Several methods of handling materials will be proposed and evaluated to determine the amount of human and/or robotic labor required for each procedural step. Simulations will be start to finish, that is, from seed (or plant propagules) through all the steps necessary to obtain food and provide seed as initial conditions for the next crop. The model will include transport of materials from one area of the HRTF to another. Examples include spacing of pots, and subsequent return of unused container-plants, and carrying of plant materials from the growth area to the processing and storage areas.

Once the models are completed, mission scenarios will be simulated to determine which proposed processes lead to the best system performances. Indices of ALSS performance will include:

1. Number of astronauts supported,
2. Manual labor requirements,
3. Energy consumption,
4. Minimum reserves for life support elements,
5. Flexibility,
6. Adaptability,
7. Complexity,
8. Reliability,
9. Storage volume,
10. Maintainability,
11. Survivability,
12. Launch mass, and
13. Launch volume.

Obviously, the number of people an ALSS can support, the manual labor requirements, and energy consumption are important characteristics. Some designs will have more reserve capacity than others, and higher marks should be given to those with the larger reserves of food and other life support elements. The ability to grow more of some crops and less of others (flexibility) is important, as astronaut's taste preferences may change over time. The ability to quickly adapt to changes in mission scenarios due to unforeseen events is an important feature of an ALSS and of the MSAT models and simulation. Simplicity usually means a lower probability of failure and larger chance of completing the mission successfully. Simplicity also affects the maintenance and/or repair of machinery used to automate materials handling in an ALSS. Storage volume may be viewed as positive or negative. In some cases, large storage volumes may be linked to large reserve stores. In other cases, the large storage volume may be required because some components require a long time to recycle. Thus, the index of merit may be the total volume required per person, although that is obviously nonlinear and dependent on the number of astronauts supported. The ability of the ALSS to maintain its functionality in a survival mode, without crew, is an especially important feature. Finally, one cannot ignore the costs of getting an ALSS into space; thus, launch criteria such as mass and volume cannot be ignored.

Mission scenarios which are infeasible for all modeled processes will be identified as high priority areas for further experimental research to discover more efficient and effective techniques.

5. Animated Graphics Simulation

Once the "what to do" steps have been defined by the operations simulations, the question of "how-to-do-it" remains. This will be answered by developing 3-D graphical models of the HRTF and by simulating the materials handling as performed by people and mechanisms, including robots. Unlike the operations simulations, these simulations will produce realistic views of the processes and permit users to ascertain that space is available to perform the tasks and to store the machinery when not in use.

These models will include grippers and sensors which enable a robot to perceive changes in the environment and adapt to them. Collisions between the mechanisms and the HRTF structure will be identified by the simulations, and additional code will be added to avoid the problem with the real-world robots. Software tools available in JSC's Integrated Graphics Operations Analysis Laboratory (IGOAL) will be used to develop the 3-D graphical models and perform simulations of the materials handling processes which will include seeding, transplanting, transport of containers, monitoring of crop health, spacing of containers, gathering, separation of edible biomass, processing, storage, and food processing.

Results will be automatically entered into the ALSS knowledge base and compared to information from the operations and biophysical simulations to determine which combination of crops, production, and materials handling technologies work best as a complete life support system. The data from the graphical simulations will include reach and clearances, cinemas from multiple viewpoints (including some from potential machine vision camera positions and robot vision viewpoints), and kinematic analysis of manipulator actions, including timing of movements. These data are necessary to design the servos needed to move the materials in the allocated time, to determine the number of robots necessary, and to determine what each robot should do. In some cases, robots might be programmed to perform similar tasks in parallel, but in other cases, it could be preferable to have a robot perform a different task in series with other machines. These simulations will also help determine what tasks should be automated by dedicated, hard-engineered mechanisms and which should be performed by flexible, programmable machines (robots). For example, should a robot harvest wheat by reaching and removing a few grain heads? Or, should it harvest an entire container of plants and place the accumulated heads in a mechanism which threshes and separates the kernels from the straw? Is the advantage of having only the head to thresh outweighed by the difficulty of locating and removing each head of wheat? Is the advantage of working with known processes, such as threshing, outweighed by the costs associated with the limited-purpose device? Should there be a mini-combine or separate harvester and thresher, and should the plants be moved to the harvester or the harvester moved to the plants? Until the simulations are performed and the results quantified, one guess is as good as another.

6. Controlled Environment Agriculture

In controlled environment agriculture, which is a several-billion-dollar-per-year business in the U.S., intelligent robotics are needed (Simonton, 1990 Trans.) to keep prices competitive. Market forces are compelling greenhouse operations, which are labor-intensive, to automate. A major motivation is for U.S. producers to improve productivity to deal with global competition (Carney, 1988).

The greenhouse production of bedding plants and the cell/tissue culture industry could readily benefit from the development of systems engineering tools and robotics for plant materials handling. Tissue cultured plants typically cost $0.50 to $1, which limits the market to high value, horticultural crops such as ornamental flowers. If, through automation, the plants could be produced for $0.05 or less, the market would swell to include many field crops, such as strawberries. This would permit plants with genetically engineered characteristics, such as plant-host resistance to pests, to be grown commercially without the need for chemical pesticides. Even though many such genetically engineered plants exist, there is no current method of propagating the technology at an affordable price.

Even cell/tissue culture research would benefit from robotics. Imagine what would happen if a lab technician who normally performs 40,000 operations every six months in order to find a single plant with potential to be genetically superior has his/her work assisted by robots which perform 40,000 operations every day. That has the potential of accelerating research discoveries by 150 or more. What might have taken 10 years to discover would then take a month or so.

There has been extensive research and development of robotics for planting, seeding, transplanting, cutting and grafting, tray stacking and destacking, and other robotic applications in controlled environment agriculture conducted in the U.S., Canada, Great Britain, Netherlands, Germany, France, Israel, Japan, China, and Taiwan. Many of these are documented in three recent international symposia (Kurata and Kozai, 1992; American Society of Agricultural Engineers, 1991; and Kozai, 1988).

7. Field Applications of Agricultural Robots

From the cell/tissue culture lab, through the greenhouse, and to the field, the world awaits the development of affordable, reliable robotic machinery for handling plant materials. Because greenhouse growers have discovered that plants require different rooting and canopy environments, bedding plants are grown in different size and shape containers. Thus, any mechanism designed for transplanting must be flexible and programmable to accommodate these differences in size and shape of containers.

Robotics are an obvious choice for automating these delicate, complex materials handling problems. Commercial efforts are limited to transplanting of bedding plants (Beam, et al., 1991). University research efforts span a wider effort. Purdue University (Miles and Kutz, 1991; Kutz, et al., 1987; Kutz and Miles, 1986; and Kutz, et al., 1986), Rutgers University (Ling, et al., 1990; Ting, et al., 1990; and Ting, et al., 1988), and the University of Georgia (Simonton, 1990) have studied robotic transplanting and plant materials handling. Robotic harvesting of fruit has been studied in France (Grand d'Esnon, et al., 1987), in Japan (Kawamura, et al., 1986), and in Florida (Harrell, 1987; and Slaughter and Harrell, 1987). A joint research effort between Purdue University (Indiana), the Volcani Institute (Israel), and the Weizmann Institute (Israel) has developed a robot capable of selectively harvesting muskmelons (Edan and Miles, 1993; Benady and Miles, 1992; Benady, et al., 1992; Edan and Miles, 1992; and Edan, et al., 1992). In this project, fruit location is determined by a machine vision and image processing system in association with a laser plane projector which casts a beam on the melons to provide 3-D information on position. Color and aromatic volatile gases released during the ripening process are used to determine which fruits are ripe enough to harvest. Clam shell, ring type grippers grasp each fruit and carry it to a conveyor to be packaged. Field testing of the prototype in Israel by the Volcani Institute has been successful, but much work remains to make the robot flexible enough to harvest other crops and to perform other functions such as transplanting in an economically-viable fashion.

More than any other single factor, the high cost of robotic components prevents more widespread use of agricultural robotics. The lack of research funding to develop affordable sensors and servo-mechanisms for agricultural applications contributes to this malaise. NASA's funding of robotic materials handling research for an ALSS would certainly spin-off robotic technologies for agriculture.

The agricultural arena provides exceptional challenges in dealing with the mixes of mobility and manipulation considerations which need to be taken into account in creating and designing the next generation of robotic equipment.

8. Summary and Conclusions

The development of models and their use to simulate the performances of an ALSS would help focus experimental research efforts, provide data essential for design of a HRTF, and assist the spin-off of robotic materials handling technologies to agriculture. Simulations of the JSC HRTF will include biophysical processes, operations, and cinemas of plant materials handling by humans and robotic machinery. Data from models of each crop will be structured and dynamically linked to shared knowledge bases.

The development of robots for plant materials handling in an ALSS will have immediate applications in the bedding plant and cell/tissue culture industries. Spin-offs to agriculture would be numerous and highly beneficial.

9. References

American Society of Agricultural Engineers, 1991. Automated Agriculture for the 21st Century. Proceedings of Symposium held in Chicago, IL. ASAE Publication 11-91. The ASAE, St. Joseph, MI.

Averner, Maurice. Controlled Ecological Life Support System In Lunar Base Agriculture: Soils for Plant Growth.

Beam, S. M.; Miles, G. E.; Treece, G. J.; Hammer, P. A.; Kutz, L. J.; and Richey, C. B. Robotic Transplanting: Simulation, Design, Performance Tests. Paper No. 91-7027. The ASAE, St. Joseph, MI.

Benady, Meny and Miles, Gaines E. Locating Melons for Robotic Harvesting Using Structured Light. Paper No. 92-7021. The ASAE, St. Joseph, MI.

Benady, Meny; Simon, James E.; Charles , Denys J.; and Miles, Gaines E. Determining Melon Ripeness by Analyzing Head Gas Emissions. Paper No. 92-6055. The ASAE, St. Joseph, MI.

Benady, Meny; Edan, Yael; Hetzroni, Amots; and Miles, Gaines E. Design of a Field Crops Robotic Machine. Paper No. 91-7028. The ASAE, St. Joseph, MI.

Cardenas-Weber, M.; Stroshine, R. L.; Haghighi, K.; and Edan, Y. Melon Material Properties and Finite Element Analysis of Melon Compression with Application to Robot Gripping. Transactions of the ASAE. 34(3): 920-929.

Carney, L. 1988. Oglevee Products, McDonough, GA. As cited by Simonton, 1990, Trans.

Edan, Y. and Miles, G. E. 1993. Design of an Agricultural Robot for Harvesting Melons. Transactions of the ASAE. 36(2): 593-603.

Edan, Y.; Haghighi, K.; Stroshine, R.; and Cardenas-Weber, M. 1992. Robot Gripper Analysis: Finite Element Modeling and Optimization. Applied Engineering in Agriculture. 8(4): 563-570.

Edan, Yael and Miles, Gaines E. Animated, Visual Simulation of Robotic Melon Harvesting. Paper No. 89-7612. The ASAE, St. Joseph, MI.

Edan, Yael; Benady, Meny; and Miles., Gaines E. Economic Analysis of Robotic Melon Harvesting. Paper No. 92-1512. The ASAE, St. Joseph, MI.

Erickson, Jon D. and Lawler, Dennis. 1994. Integrated Computer Aided Concurrent Engineering. Dual-Use Space Technology Transfer Conference and Exhibition, NASA Johnson Space Center, Houston, TX.

Erickson, Jon D.; Aucoin, P. J., Jr.; and Dragg, J. L. 1992. A Tool for Simulating Space Exploration Initiative Operations as a Means of Obtaining Intelligent System Requirements. Proc. SPIE 1829 Conf. on Cooperative Intelligent Robotics in Space III, Boston, MA. pp.482-494.

Grand d'Esnon, A.; Pellenc, R.; Rabatel, G.; Journeau, A.; and Aldon, M. MAGALI: A Self-Propelled Robot to Pick Apples. Paper No. 87-0037. The ASAE, St. Joseph, MI.

Harrell, R. 1987. Economic Analysis of a Robotic Citrus Harvester in Florida. Transactions of the ASAE. 30(2): 298-304.

Henninger, Donald L. and Tri, Terry O. 1993. Controlled Ecological Life Support Systems Human-Rated Test Facility: An Overview. 23rd International Conference on Environmental Systems.

Kawamura, N.; Namikawa, K.; Fujiura, T.; and Ura, M. 1986. Study on Agricultural Robot. Memoirs of the College of Agriculture, Kyoto University, Kyoto, Japan. No. 129, pp. 29-46.

Kozai, T. (Ed.). 1988. High Technology in Protected Cultivation. Proc. of Symposium held in Hamamatsu, Japan. Acta Horticulturae, Vol. 230.

Kurata, K. and Kozai, T. (Eds.). 1992. Transplant Production Systems. Proc. of Symposium held in Yokohama, Japan. Kluwer Academic Publishers, Dordecht, Netherlands. Also, papers presented at this symposium are in Acta Horticulturae, Vol. 319.

Ling, Peter P.; Tai, Y. W.; and Ting, K. C. Vision Guided Robotic Seedling Transplanting. Paper No. 90-7520. The ASAE, St. Joseph, MI.

Miles, G. E. and Kutz, L. J. 1991. Robots in Transplanting. IN: Y.P.S. Bajaj, Ed. Biotechnology in Agriculture and Forestry, Vol. 17 High-Tech and Micropropagation I. Springer-Verlag, Berlin. pp.452-468.

Ming, D. W. and Henninger, D. L. (Eds.). 1989. Lunar Base Agriculture. Crop Science Society of America, Inc.; American Society of Agronomy, Inc.; and Soil Science Society of America, Inc. 677 S. Seagull Road, Madison, WI.

Petri, D. 1993. Program Life Cycle and the System Engineering Process. JSC-49037. NASA Johnson Space Center, Houston, TX.

Schwartzkopf, Steven. 1991. Lunar Base Controlled Ecological Life Support System (LCELSS): Preliminary Conceptual Design Study. LMSC-F280196. Lockheed Missiles and Space Company, Sunnyvale, CA.

Slaughter, D. C. and Harrell, R. C. 1987. Color Vision in Robotic Fruit Harvesting. Transactions of the ASAE 30(4): 1144-1148.

Simonton, W. Robotic End-Effector for Handling Greenhouse Plant Material. Paper No. 90-7503. The ASAE, St. Joseph, MI.

Simonton, W. 1990. Automatic Gernium Stock Processing in a Robotic Workcell. Transactions of ASAE, Vol. 33, No. 6, pp.2074-2080.

Ting, K. C.; Yang, Y.; and Fang, W. 1990. Stochastic Modeling of Robotic Workcell for Seedling Plug Transplanting. Paper No. 90-1539. The ASAE, St. Joseph, MI.

Ting, K. C.; Giacomelli, G. A.; and Shen, S. J. 1986. Robotic Work Cell Development for Seedling Transplanting. Paper No. 88-1027. The ASAE, St. Joseph, MI.

AIAA-94-1260-CP

ROBOTIC HAULING TRUCK FOR SURFACE MINING

Keith Chrystall, Patrick Feighan, Peter Wojcik
Alberta Research Council, Advanced Computing and Engineering
6815 - 8 Street NE, Calgary, Alberta T2E 7H7
ph: (403) 297-2600, fax: (403) 297-2339
(kgc, feighan, peter)@skyler.arc.ab.ca

Julian Coward
Syncrude Canada Ltd., Edmonton Research Center
10120 - 17 Street, Edmonton, Alberta T6P 1V8
ph: (403) 449-2800, fax: (403) 449-2805

Ron Eirich, Clement Laforce
Defence Research Establishment Suffield,
Box 4000, Medicine Hat, Alberta T1A 8K6
ph: (403) 544-4733, fax: (403) 544-3761
claforce@dres.dnd.ca

Abstract

The paper describes the concept and design considerations of a robotic hauling truck system intended for use in surface mine applications. An engineering prototype of the designed system will be implemented at the Syncrude oil sands mine near Fort McMurray in northern Alberta. The robotic truck will perform routine hauling functions through a combination of manual, teleoperated and autonomous activities. The primary objective for the design is to provide an integrated intelligent vehicle system that is capable of navigating autonomously across outdoor terrain avoiding collisions with obstacles that could be encountered. This capability will enable the robotic hauling trucks at Syncrude to operate autonomously on the routes between the loading area and dumping area. Operations at the loading and dumping areas will be performed using manual or teleoperated control. Future developments will allow autonomous operation of the robotic truck at the load and dump areas.

1. Introduction

The semi-autonomous robotic truck system project is being pursued by a research team consisting of members from Syncrude Canada Ltd., Defence Research Establishment Suffield and the Alberta Research Council.

Syncrude Canada Ltd. (Syncrude) operates a world scale oil sands mining, extraction and upgrading operation near Ft. McMurray, in northern Alberta. In 1992, about 50 million cubic metres of overburden were removed to expose the oil sands ore. Currently thirty 170 ton hauling trucks are used 24 hours per day to remove overburden. Within four years it is estimated that fifty hauling trucks will be necessary, with the balance of new trucks having a 240 ton capacity. Syncrude is considering automating the operations of its hauling truck fleet in order to reduce the mine operating costs.

Copyright © 1994 by Government of Canada, Department of National Defence. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

Defence Research Establishment Suffield (DRES) has a well established robotic vehicle development program. Land, air, and sea based vehicles have been successfully developed for applications including live fire target transport, surveillance, hazardous material handling and mine detection. An important result from this work is the development of ANCÆUS, an advanced and proven generic remote vehicle control system. ANCÆUS is an inexpensive semi-autonomous vehicle control technology able to interface with virtually any surface vehicle as well as with their payloads.

The Alberta Research Council (ARC) is the oldest and largest provincial research organization in Canada. The ARC is experienced in leading and participating in technology development and technology transfer projects in many fields including autonomous systems, robotics and advanced computer applications.

The intent of the combined research team is to transfer the military robot vehicle technology to a prototype robotic truck application in Syncrude's mine. The following sections of the paper describe the application in detail, the technical challenges and the concept for the prototype vehicle. The paper concludes with comments about the status of the development effort.

2. Current mine operation overview

Syncrude operates a large surface mining plant in a region of northern Alberta called the Athabasca oil sand deposits. These deposits are one of the largest oil reserves in the world, with almost 1000 billion barrels of oil, over four times the amount in Saudi Arabia. Of the Athabasca reserves, about 10% are recoverable by surface mining techniques, and over half could be recovered by in-situ methods.

In the Syncrude area, the oil sands average about 42 metres thick, and contain an average of 11% oil by weight. Syncrude mines and processes oil sands containing over 6% oil, with any lower grade material

being treated as waste. The oil sands are covered by varying amounts of overburden. At Syncrude the average overburden depth is about 20 metres. The mine location was chosen to minimize overburden removal required during the initial years of mining.

2.1. Oil sands surface mine operation

The first stage in the Syncrude operation is to remove trees, muskeg and any reclaimable soil material. This material is stored for later reclamation of the mined out area. The overburden is then stripped using a truck and shovel fleet.

The underlying oil sands material is mined in two stages, the first using four large draglines with 65 cubic metre buckets which excavate the oil sands and place the ore in linear piles called windrows. The draglines also reject waste bands in the oil sand, and dump this material back into the mined-out pit. In the second mining stage, the windrows are recovered by four bucket wheel reclaimers that place the oil sands onto conveyors for transportation into the crude oil producing plant. There are four trains of conveyors, with belts 1.8 metres wide, and total conveyor lengths of up to 7 kilometres. The individual conveyor flights are up to 2.5 kilometres long. In the oil producing plant, the oil sands material is appropriately processed to produce synthetic crude oil, that is finally pipelined to markets.

2.2. Hauling operations

Syncrude currently uses its 170 ton hauling truck fleet to transport overburden covering oil sands. The trucks are loaded using mining shovels, either with waste overburden or with oil sand. The trucks conveying overburden travel to a waste dump area, most of which are located in the mined out pit. The trucks dump the overburden and then return to the loading area to repeat the cycle.

The trucks carrying oil sand travel to an Auxiliary Production System (APS) crusher/feeder. The oil sand is dumped in a hopper, crushed and fed onto a conveyor system for transportation into the plant. Again, the truck returns to the loading area to pick up additional loads. Figure 1 illustrates the major functions of the hauling truck operation.

- The loading area is at the shovel or loader. The location of, and routing to this area changes frequently.
- The main loaded haul, between the loading area and the dump (or crusher). Normally, this is a relatively long section, and the routing will not change for particular load point and dump.
- The return empty haul is often, but not always, the same as the loaded haul route.
- The overburden dump area is where the waste material is dumped. The location and routing across this area changes frequently.
- The APS crusher or feeder-breaker is used for oil sand feed. This location is fixed.
- Special operations, such as fueling, maintenance, extraction rejects, moving to a new location or operation, and other non-routine tasks.

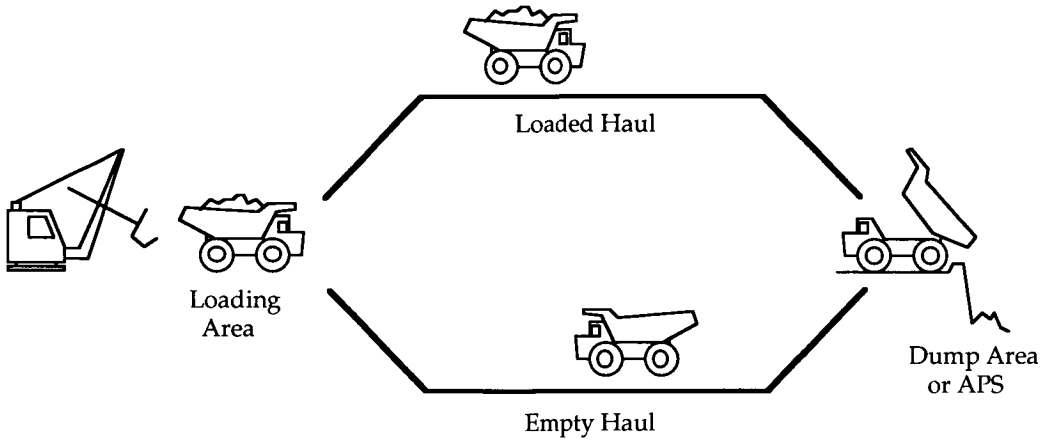


Figure 1 - Hauling Truck Operation

The degree of automation that can practically be applied to the different hauling operations will vary. The fully automated mode is easiest to implement on the loaded and return hauls. Initially, manual or tele-operated modes will be used at the shovels and dump areas. The overburden dump area is the hardest to automate because its location changes quite frequently. Special operations, such as driving into maintenance bays, would be carried out manually.

2.3. Incentives for automation

There are a number of incentives to automate the trucking operation:

- **Productivity Increases.** The use of robotic trucks could lead to higher truck utilization and thus higher productivity, as there would be less downtime due to operator breaks and no delays at shift changes.

- **Maintenance Costs.** Robotic trucks should drive more consistently, giving lower maintenance costs, and also higher availability.

- **Reduction in mine operating costs.** There is a potential in a robotic truck mining scheme to reduce the number of operators, and thus save payroll costs.

3. Robotic Truck System concept

Automating hauling truck fleet operations is based on the development of a semi-autonomous robotic vehicle that could be operated remotely. Remote operation of a driverless truck will require integration of several technologies on board the vehicle. These technologies include communication, control, guidance, obstacle detection, collision avoidance and condition monitoring. The concept of a semi-autonomous robotic truck is illustrated in Fig.2

The robotic truck system could have three modes of operation: manual, teleoperated, and fully autonomous. In the manual mode of operation a driver has full control over the vehicle and performs all functions as if no automation were implemented. When a vehicle is teleoperated, there is no driver in the cab, and the truck is driven and controlled remotely from a control station using video monitors and input devices to control speed, steering and other functions. In the autonomous mode of operation, the vehicle drives independently using a navigation system for setting route to the destination as well as an obstacle detection and collision avoidance system.

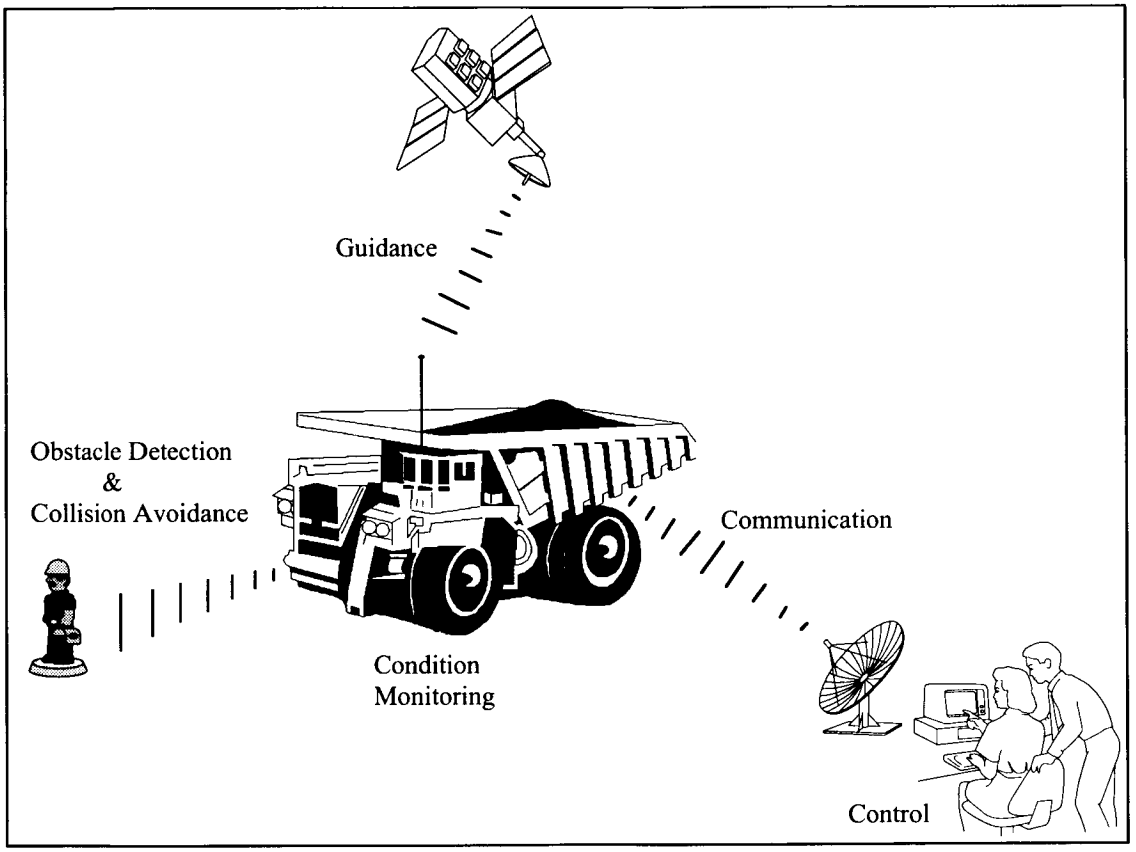


Figure 2 - Robotic Truck System Concept

Initially, it is expected that the robotic truck will be operated in manual or teleoperated mode at the load (shovel or loader) and dump (dump or crusher) areas. The robotic truck will operate autonomously on the loaded haul from the load area to the dump area, and on the empty return haul from the dump area back to the load area. Special operations (fueling, maintenance, etc.) will continue to be carried out manually.

A fully loaded truck may weigh as much as 500 tons and be capable of traveling at 35 mph. Collisions either with obstacles such as big rocks or with other vehicles would have to be avoided. All authorized vehicles in the area of truck operation will be fitted with transponders, and all automated truck haul roads will be closed to normal traffic. However, the robotic truck must also be able to avoid collisions with unauthorized equipment, personnel, large rocks and animals. This issue is discussed in more detail in Section 3.4.

The major technologies needed for the development of a semiautonomous robotic truck system are at different stages of maturity. The current status of these technologies is summarized in Table 1. A detailed discussion of enabling technologies follows the table.

TECHNOLOGY	DESCRIPTION	CURRENT STATUS
Communication	Communication between trucks and the central office, etc.	Systems are available.
Control	Overall control of the truck.	Computer control systems are available. Software development is needed.
Guidance	Guidance and location of trucks along a road.	A number of techniques have been developed, but further work is needed.
Obstacle Detection and Collision Avoidance	Avoidance of objects, people, etc. in the path of the truck.	Some sensors are available, but significant work is needed.
Condition Monitoring	Monitoring of truck health and operation.	Systems available, further development is needed

Table 1 Status of technologies needed for robotic trucks

3.1. Communications

A reliable means of communication between the truck and the location of the operator station is needed. Data, control signals, and video images from the truck all need to be transmitted. Fail-safe communications systems are also important given the potential for communication links to be compromised or broken in an outdoor environment. DRES has addressed all of these issues with the ANCÆUS robot vehicle control system using conventional radio communication systems.

3.2. Control

Automatic control of the robotic truck is essential. There will need to be features to permit a driver to over-ride automated systems and manually drive the truck when needed. The automatic control functions will need to accommodate teleoperation of the truck when required. The ANCÆUS system addresses most of these control requirements.

3.3. Guidance

When the semiautonomous truck is moving under automatic control, some form of guidance system is essential. Possible systems include:

1. Guide Wires.
2. Global Positioning System.
3. Microwave Location.
4. Lasers.
5. Inertial Systems.
6. Dead Reckoning.

Each system has its advantages and disadvantages. DRES has successfully incorporated Global Positioning System technologies into the ANCÆUS control system. Other guidance systems are commercially available. In mining applications more than one guidance system will be implemented to provide system redundancy for failsafing.

3.4. Obstacle detection and collision avoidance

Detecting obstacles in the way of the truck and avoiding collisions are critical capabilities for the robotic truck. A driverless truck cannot be considered safe to operate unless there is a rugged, reliable and fail-safe method for obstacle detection and collision avoidance. The types of obstacles that need to be considered include:

- Other automated haul trucks
- Authorized shovels, bulldozers, graders, pickup trucks etc.
- Authorized people
- Unauthorized people, animals, vehicles
- Rocks on the road dropped by other trucks
- Potholes in the roadway

The primary sensing techniques that could be used for obstacle detection include:

1. Infra-red sensors.
2. Ultra-sonic sensors.
3. Optical sensors.
4. Microwave sensors
5. Laser scanners.
6. Computer Vision systems

No one sensing technique will effectively detect all the obstacles that need to be considered. Combinations of these sensors, each detecting a range of different types of obstacles, will likely be required to handle all the anticipated circumstances.

These primary sensing systems would be directly determining the presence of obstacles in the path of the truck. These systems would be backed up by other systems such as:

7. Mechanical bumpers mounted around the truck.
8. Security approaches to prevent unauthorized entry of vehicles and people into the truck's zone of operation.
9. Emergency stop 'buttons' located at strategic places around the truck's zone of operation.
10. Transponders mounted on authorized vehicles and personnel to provide automatic identification to the truck.
11. Video images from the truck to the remote operator.

3.5. Condition monitoring

In a manually driven truck, the operator will often notice if the truck is operating abnormally by listening to the

engine, feeling any vibrations, or looking at the gauges. In an automated truck, a condition monitoring system is needed to detect problems in truck operation before any serious damage to the truck occurs.

Capabilities are required for detection of incipient problems with minimal false alarms. The information will need to be communicated to the location of the operator for processing or for maintenance call-out. Condition monitoring systems are now available, but further development will be needed to create a system that is able to detect and identify subtle changes in truck operation.

4. System design

The prototype robotic hauling truck system will consist of two major elements: truck control station and on-board equipment mounted on each truck in the fleet. The overall system is illustrated in Fig.3. The design of the prototype system is derived from previous implementations of the ANCÆUS robotic vehicle control technology developed by Defence Research Establishment Suffield.

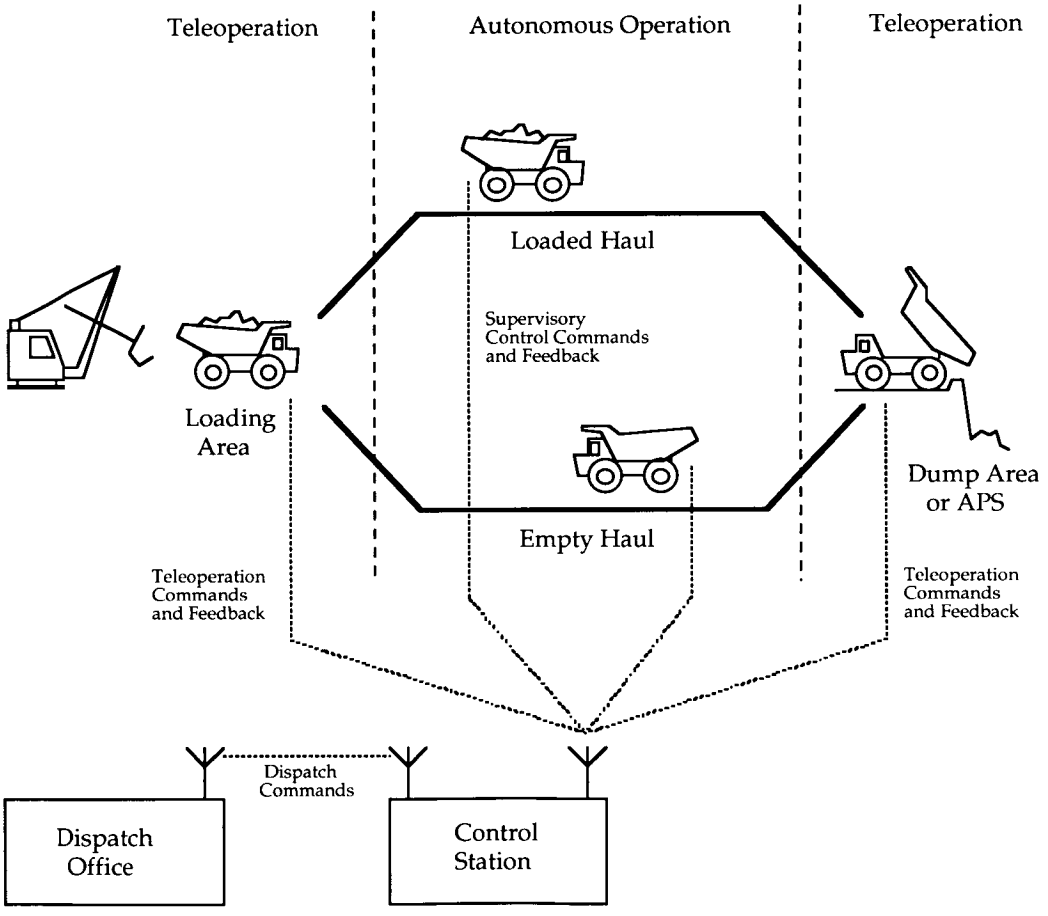


Figure 3 - Robotic Truck System

4.1. On-board subsystems

The modular nature of ANCÆUS (Fig. 4) will allow the control system to be efficiently installed on the robotic truck. Only one module needs to be redesigned to suit the control inputs and outputs of the truck. This is the only major piece of development to create the basic vehicle control system. Other ANCÆUS modules in the basic vehicle electronics package (i.e. excluding obstacle detection and collision avoidance and health monitoring) remain the same and do not require redesign.

All modules are interconnected with a high speed fibre optic data link that provides effective noise immunity. The modules communicate by passing high level commands (which remain constant for all vehicle types).

The vehicle electronics package consists of four primary modules: the vehicle control processor, the vehicle personality module, the video control module and the vehicle navigation module.

The vehicle control processor module functions as a network gateway between the control data link and the vehicle fibre optic data link. High level functions of the vehicle, such as semi-autonomous navigation, health monitoring, obstacle detection and collision avoidance would also reside in this module.

The vehicle control processor is housed in a rugged temperature-controlled weather-proof enclosure. The main processor (currently a 68030 single board computer) is plugged into the 12-slot VME back plane. As the system develops over time, more processors and/or special hardware, such as vision processors, can be added.

A novel approach to temperature control of the enclosure allows commercial grade components to function where industrial or military grade components were previously required. The temperature is maintained by using thermo-electric heat pumps, which can either heat or cool the enclosure.

Software for the vehicle control processor is written in C and speed-critical routines have been written in Assembler. The code can reside in Erasable Programmable Read Only Memory (EPROM) or can be loaded from disk.

The vehicle personality module controls all vehicle-related activities on the robotic truck such as starting and stopping the engine, steering, speed control, and braking. The personality module also monitors vehicle information such as engine temperature and revolutions per minute, charging current and voltage for the primary battery and other useful vehicle parameters.

The video module allows up to 256 video and sound inputs to be switched to up to four different RF video transmitters and/or fibre optic video channels. As well as switching video and sound, each channel can have an associated pan, tilt, zoom and flood lamp control channel.

All functions associated with determining the truck's position reside in the navigation module. Any module requiring navigation data can request it from this module. The current ANCÆUS system uses a Trimble GPS receiver to obtain global positioning data as well as pitch and roll sensors to alert the vehicle to dangerous attitudes. The receiver can also make use of differential GPS data to increase its accuracy. The specific navigation technology to be used for the truck has yet to be determined.

The application modules connect to the vehicle fibre optic data link in the same manner as the four primary modules. Application modules are required to implement any functions not included in the primary four modules. A typical function for an application module may be the control of a robotic arm. Currently, no application modules are projected for the prototype robotic truck.

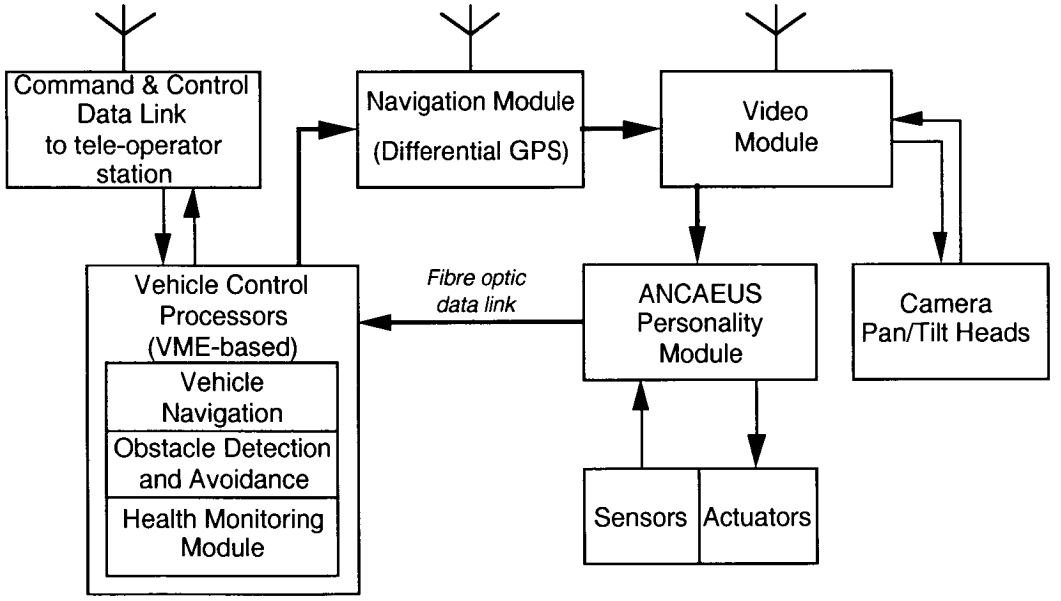


Figure 4 - On-board subsystems

4.2. Control station

The control station (Fig.5) for the robotic truck is modular in nature, consistent with the design philosophy of the ANCÆUS system. The design allows different control consoles to be connected as well as efficient interfacing for a given application. It is anticipated that the control station for the truck will transmit data to and from the vehicles through a RF modem. The control station is further composed of three modules: the control station computer, video control module and the control console module.

4.3. Additional Features of ANCÆUS

The primary ANCÆUS design goals are flexibility and resistance to obsolescence. To this end the system is modular, which allows it to be easily upgraded. The system also can be placed on most vehicle chassis with a minimal amount of redesign due to the unique personality module concept discussed above. The personality module accepts high level commands from the control station and implements these commands for a particular vehicle.

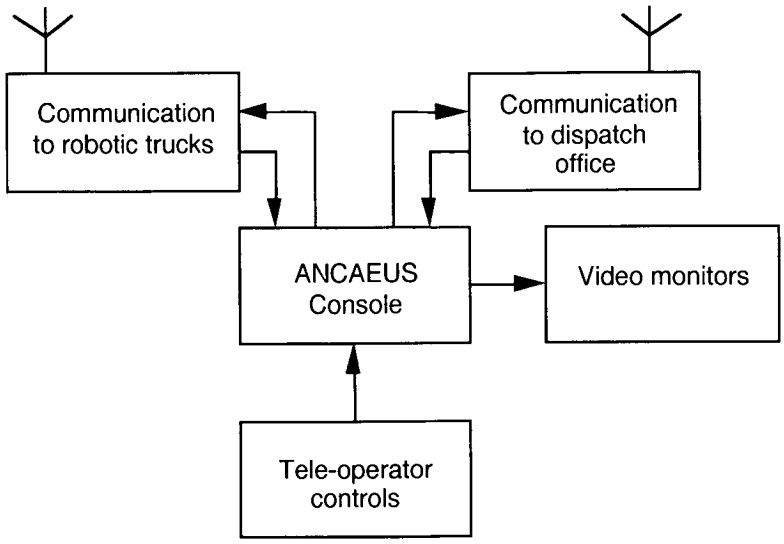


Figure 5 - Control station

The ANCÆUS system also has the following features:

- Up to 254 vehicles can be addressed on one network.
- Up to 255 hardware modules (personality or application) can be installed on each vehicle.
- Hierarchical vehicle safety built into network.
- RF data link repeater vehicles are possible.
- Modular vehicle system:
  - uses onboard fibre optic data link for noise immunity;
  - allows quick replacement of faulty hardware,
  - allows modules to be placed close to controlled activity;
  - allows easy installation of application specific modules;
  - distributed processing increases vehicle processing power;
- Uniform control code structure.
- Application control codes require no software changes.
- Temperature-controlled Vehicle Control Processor uses commercial components in adverse environmental conditions
- Built-in video management
- Can be efficiently installed on most vehicle chassis.

5. Related work

There has been considerable work done on automation of both surface and underground mining. In North America, there appears to be more automation progress in underground mining than in surface mines since, in some underground mines, safety aspects give large incentives to develop driverless vehicles. The following are some of the key robotic truck research activities that have been identified and have relevance to this project:

- **INCO** has developed an automated 70 ton haulage truck (AHT) that uses an "I" beam on the roof of a mine tunnel for guidance [1]. The AHT can haul, dump and return automatically, but to date some manual input is required for loading. The overall project started in 1983, and the AHT was commissioned in 1989, and has been running successfully since then. INCO is also working on an automated LHD vehicle, that is loaded using teleoperation, travels along a tunnel automatically, and is dumped manually [5]. The LHD is guided by a reflective strip in the ceiling of the tunnel. Tests of the LHD have been quite successful.

- **Noranda** As a joint project with INCO, Noranda is working on an automated LHD vehicle that can muck automatically. This LHD is fitted with many sensors to determine the vehicle health.

- **Potash Corporation of Canada,** have automated a continuous borer type mining machine. A large number of sensors are used to detect ore grade and machine health. A laser beam is used to provide guidance in the tunnel. The machine can be operated in fully automatic, semi-automatic or in manual modes.

- **Defence Research Establishment Suffield (DRES),** have developed an ARGO 8 wheel ATV for full automation and tele-operation [2]. The system was developed in a four month period to detect land mines. The control and sensor package was developed to be very rugged (near military specifications) and to be largely vehicle independent, with only one system requiring change to fit on another type of vehicle (such as a mining truck). No collision avoidance sensors were installed [8].

- **Caterpillar** is developing a robotic vehicle. Little information is known about this work since it is proprietary to Caterpillar.

- **BHP** in Australia has carried out work on automation of different mining equipment.

- **Kamatsu** of Japan is researching a robotic truck but, according to the literature, the work does not appear to be as advanced as the DRES ANCÆUS system.

- **The Alberta Research Council** (ARC) has been developing technology to support control of robotic devices in the presence of low communication bandwidths for the Canadian Space Agency [7]. This technology could be applied to the control of multiple robotic vehicles.

- **PRECARN Associates Inc.** through its ARKS, IRIS, and IGI projects has made several technology developments that are applicable to the semiautonomous mining truck [3]. The proposed PRECARN MAP project also has objectives that are similar to the semiautonomous truck development and there is potential for a significant degree of synergy between the two projects.

- **The National Research Council of Canada** has developed a number of technologies that are relevant to this study [4]. They include semi-autonomous robots, computer vision systems and other sensor developments.

- **CMU** (Carnegie Mellon University, Pittsburg) has developed and tested a navigation system (FASTNAV) implemented on a 150 ton Caterpillar dump truck [6].

6. Current Status

The robotic truck project is currently in the initial stages of development. Research related to collision detection and obstacle avoidance systems has begun and preliminary results are expected by mid-summer 1994. By fall of 1994 a detailed design concept for the prototype robotic truck is expected to be ready and an operation prototype ready for testing by winter 1995. In parallel with these technical developments the project team will also be examining the issues of integrating a robotic truck fleet into routine mine operations and developing strategies for bringing this technology to the international market.

References

1. Baiden, G.R. Automatic Haulage Truck - Design, Development and Mine Implementation at INCO Ltd. Paper #24, CIM conference, April 26, 1992, Montreal.

2. Flakstad, N. Military Research Moves to meet Mine Menace, The PEGG (APEGGA newsletter), May, 1993.

3. IRIS (the Institute for Robotics and Intelligent Systems), A network of Success, Publication by PRECARN, undated.

4. NRC, The Institute for Information Technology, Information brochures, and personal communication with S. Elgazzar, May, 1993.

5. Baiden, G.R. Combining Teleoperation with vehicle guidance for improving LHD productivity at INCO Limited. Paper presented at the CIM meeting in Calgary, May, 1993.

6. Hemani, A., Robotics and Automation in Mining, IEEE Canadian Review - Fall 1993, pp. 11-15.

7. Feighan, P., Chrystall, K., Dagnino, A, Knowledge Based Task Planning for Supervisory Control of Telerobotic Systems , Proceedings of the 1993 DND Workshop on Advanced Technologies in Knowledge Based Systems and Robotics, Nov. 14-17, 1993, Ottawa, Ontario.

8. Eirich, R., Kramer, A., A Generic Semi-autonomous Ground Vehicle Control System , Proceedings of the 3rd Conference on Military Robotic Applications - Military Robotic Vehicles MRV91, Sept. 9-12,1991, Medicine Hat, Alberta, pp. 68-73.

AIAA-94-1261-CP

A Nonlinear Strategy for Sensor Based Vehicle Path Control

R. Mayr

Institute of Robotics Research, University of Dortmund
P.O.Box 50 05 00, W-4600 Dortmund 50, FRG
or
Center for Advanced Transportation Technologies,
University of Southern California,
3740 McClintock Avenue. Los Angeles, CA 90089-2563

Abstract

A new method to stabilize a vehicle, which is autonomously guided on the road by an optical sensor, is presented. Since the dynamical behavior of a vehicle is nonlinear and contains mutual couplings in the state variables, the design especially of algorithms for transverse control is based on the universal valid principle of nonlinear decoupling and control. This method works independent from any operating point and thus a high degree of accuracy and free pole assignment for the overall system is provided.

1 Introduction

In the last years many efforts have been achieved in realizing and integrating automation components in motor vehicles. In this context the task is followed to support the driver by efficient kinds of control and safety systems [1]. In this way anti block systems, methods to prevent a vehicle from skidding on icy roads [2] as well as the application of collision avoidance strategies are described [3]. Besides a very important aspect to increase safety in traffic is to direct a vehicle equipped with an on-board system automatically on the road [4]. Here a method is described, where the geometry of the street is detected by an optical system mounted on the car. Based on the information coming from this sensor the control system provides adequate control signals to stabilize the vehicle on the right lane. As the street has to be followed exactly, a high quality control system for the vehicle is required. In this paper a very efficient strategy of transverse control based on nonlinear formulations is presented. Furthermore a longitudinal controller to influence the velocity of the vehicle is described.

Copyright ©1994 by R. Mayr. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission

2 Mathematical model of the vehicle

As already mentioned, the task is followed to develop a control system, which directs the vehicle automatically along the road under consideration of a nominal speed. The right edge of the street is traced by an optical sensor, e.g. a video camera. In this way the distance l_{tr} between the point P on the optical axis and the point E on the edge of the road can be determined. By further consideration of distance l_{lo} between point P and the center of gravity of the vehicle a proportion of the car's direction of movement in dependence of the course of the road can be indicated. This structure is illustrated by Fig. 1, where especially in the lower sketch the kinematics of the vehicle including the optical sensor is shown. Please note that l_{lo} represents a constant parameter and, in contrast, l_{tr} is a dynamic variable.

To develop an efficient control system a mathematical model describing the dynamical behavior of the vehicle is a very important supposition. Besides the requirements of high accuracy this model, which consists of several differential equations, should be of certain clearness. A single track model describing transverse and longitudinal dynamics, neglecting roll and pitch angles and comprising front and rear wheels to one fictitious wheel respectively is well suited [5]. It is transferred into state space description, which will be

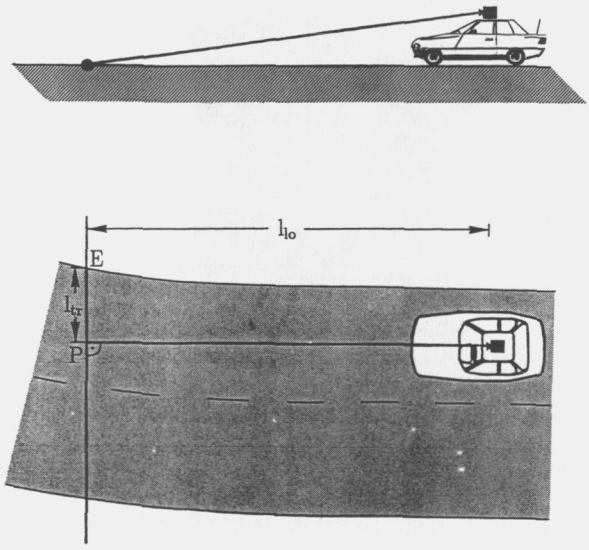


Figure 1: Mode of operation of the sensor

the starting point for the design of the path control system:

$$\dot{\underline{x}} = \begin{bmatrix} x_3 + \frac{1}{m}\frac{T(\underline{x})\,x_1}{x_4} - \frac{1}{m}\frac{S_h(\underline{x})}{x_4} \\ x_3 \\ -\frac{l_h}{\theta}\,S_h(\underline{x}) \\ -\frac{1}{m}\,T(\underline{x}) \end{bmatrix} + \begin{bmatrix} -\frac{1}{m\,x_4} & -\frac{1}{m}\frac{x_1}{x_4} \\ 0 & 0 \\ \frac{l_v}{\theta} & 0 \\ 0 & \frac{1}{m} \end{bmatrix} \underline{u}(t) \tag{1}$$

$$\underline{y} = \begin{bmatrix} x_2 \\ x_4 \end{bmatrix}.$$

The 4-dimensional state vector $\underline{x}(t)$, which consists of the dynamical variables, as well as the 2-dimensional input and output vectors $\underline{u}(t)$ and $\underline{y}(t)$ are introduced as follows:

$$\begin{aligned} \underline{x} &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T \\ &= \begin{bmatrix} \beta & \psi & \psi' & v \end{bmatrix}^T \\ \underline{u} &= \begin{bmatrix} u_1 & u_2 \end{bmatrix}^T = \begin{bmatrix} S_v & H \end{bmatrix}^T \\ \underline{y} &= \begin{bmatrix} y_1 & y_2 \end{bmatrix}^T = \begin{bmatrix} \psi & v \end{bmatrix}^T. \end{aligned} \tag{2}$$

Inputs are front side force S_v of the vehicle depending on the steering angle δ_v by

$$S_v(\underline{x}) = c_v \left(x_1 - \frac{l_v}{x_4}\, x_3 + \delta_v \right) \tag{3}$$

and rear driving or braking force H, while output variables are the yaw angle ψ describing the orientation of the vehicle and the velocity v of the car. As shown in Fig. 2, important state variables are the yaw angle ψ, the yaw velocity ψ',

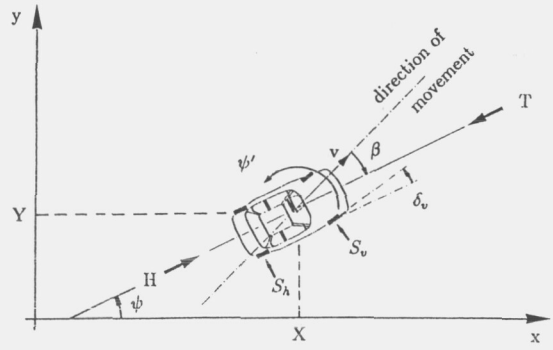


Figure 2: Dynamical variables of the vehicle

which consists its first derivation, the longitudinal velocity v and the sideslip angle β. The vehicle is of mass m and of moment of inertia θ, while c_v, c_h, l_v and l_h are further constant parameters. The variables $S_h(\underline{x})$ and $T(\underline{x})$, which consist of the rear side force and the air resistance respectively, depend on their turn on the state variables and can be computed by

$$\begin{aligned} S_h(\underline{x}) &= c_h \left(x_1 + \frac{l_h}{x_4}\, x_3 \right) \\ T(\underline{x}) &= c_w \frac{\varrho_L}{2} A\, x_4^2. \end{aligned} \tag{4}$$

Aerodynamic resistance coefficient, atmospheric density and surface of the vehicle's cross section are represented by the parameters c_w, ϱ_L and A. To keep clearness of the model equations for the derivation of the control laws these variables are not inserted explicitly. In this way also the front side force is stipulated as an input variable of the model.

Furthermore the actual position coordinates X and Y of the center of gravity in the stationary system is evaluated

with respect to the differential equations

$$\begin{aligned} \dot{X} &= v\ \cos(\psi - \beta) \\ \dot{Y} &= v\ \sin(\psi - \beta). \end{aligned} \tag{5}$$

However, the coordinates X_P and Y_P of the point P on the optical axis, which is of distance l_{lo} to the center of gravity are determined by

$$\begin{aligned} X_P &= X + l_{lo}\ \cos\psi \\ Y_P &= X + l_{lo}\ \sin\psi. \end{aligned} \tag{6}$$

Please note that equations (5) and (6) are only required for the simulation. As shown in section 3, however, they are not required to develop the control algorithms.

3 Nonlinear Path Control

3.1 Design of the control algorithms:

The dynamical behavior of a vehicle is extremely nonlinear and consists of mutual couplings. Control laws, which are based on a linearized model, would have the disadvantage of dependence from an operating point. However, if the design of the control laws is based on the universal method of nonlinear decoupling and control, all couplings and nonlinearities are compensated in a direct way and the controlled vehicle will be impressed a linear behavior with free pole assignment [6]. Based on state space description (1) the task is to find algorithms to control the vehicle's yaw angle ψ and the velocity v by influencing front side force S_v and rear longitudinal force H.

The elements $C_1(\underline{x})$ and $C_2(\underline{x})$ of the matrix $\underline{C}(\underline{x})$, representing the coordinates of the car's center of gravity, are related to a subsystem each [7]. For both subsystems first the differential orders, which determine the order of the output variables' derivations connected in a direct way to an input variable, have to be found out respectively. To determine the value greater zero for the subsystem i the differential operator $N_A^k C_i(\underline{x})$ with $k = 0, 1, 2, ...$, which can be recurrently evaluated by

$$N_A^k C_i\ (\underline{x}) = \Big[\frac{\partial}{\partial \underline{x}}\ N_A^{k-1} C_i(\underline{x})\Big] \underline{A}\ (\underline{x}) \tag{7}$$

under consideration of the initial value

$$N_A^0 C_i\ (\underline{x}) = C_i(\underline{x}), \tag{8}$$

is introduced. Hence the differential orders of the subsystems are evaluated by the following formulation:

$$d_i = \min\Big\{ j : \Big[\frac{\partial}{\partial \underline{x}}\ N_A^{j-1} C_i\ (\underline{x})\Big]\ \underline{B}(\underline{x}) \neq \underline{0}^{\mathrm{T}} \quad j = 1, 2, \ldots, n \Big\} \tag{9}$$

with $\underline{0}$ as the zero vector. With respect to state space description (1) the differential operators $N_A^k C_i(\underline{x})$ as well as the parts $\Big[\frac{\partial}{\partial \underline{x}}\ N_A^k C_i(\underline{x})\Big]\underline{B}\ (\underline{x})$ are evaluated for subsystem $i = 1$ to

$$\begin{aligned} N_A^0 C_1(\underline{x}) &= x_2 \\ \Big[\frac{\partial}{\partial \underline{x}}\ N_A^0 C_1(\underline{x})\Big]\underline{B}\ (\underline{x}) &= \begin{bmatrix} 0 & 0 \end{bmatrix} \end{aligned} \tag{10}$$

and

$$\begin{aligned} N_A^1 C_1(\underline{x}) &= x_3 \\ \Big[\frac{\partial}{\partial \underline{x}}\ N_A^1 C_1(\underline{x})\Big]\underline{B}\ (\underline{x}) &= \Big[\frac{l_v}{\theta} \quad 0\Big]. \end{aligned} \tag{11}$$

Furthermore for subsystem $i = 2$ these parts yield

$$\begin{aligned} N_A^0 C_2(\underline{x}) &= x_4 \\ \Big[\frac{\partial}{\partial \underline{x}}\ N_A^0 C_2(\underline{x})\Big]\underline{B}\ (\underline{x}) &= \Big[0 \quad \frac{1}{m}\Big]. \end{aligned} \tag{12}$$

As for subsystem $i = 1$ part $\Big[\frac{\partial}{\partial \underline{x}}\ N_A^0 C_1(\underline{x})\Big]\underline{B}\ (\underline{x}) = \underline{0}^{\mathrm{T}}$ but part $\Big[\frac{\partial}{\partial \underline{x}}\ N_A^1 C_1(\underline{x})\Big]\underline{B}\ (\underline{x}) \neq \underline{0}^{\mathrm{T}}$ and for subsystem $i = 2$ already part $\Big[\frac{\partial}{\partial \underline{x}}\ N_A^0 C_2(\underline{x})\Big]\underline{B}\ (\underline{x}) \neq \underline{0}^{\mathrm{T}}$ the differential orders d_i for both subsystems result in

$$\begin{aligned} d_1 &= 2 \\ d_2 &= 1\ . \end{aligned} \tag{13}$$

These values for the differential orders signify the immediate effect of the system inputs on the second derivation of the yaw angle and the first derivation of the car's velocity. As the output variables consist of the yaw angle, which is of kinematic character, and the velocity representing a kinetic variable, one receives different values for the differential orders of each subsystem.

For further consideration also the operators

$$\begin{aligned} N_A^2 C_1(\underline{x}) &= -\frac{S_h\ l_h}{\theta} \\ N_A^1 C_2(\underline{x}) &= -\frac{c_5}{m} x_4^2 \end{aligned} \tag{14}$$

have to be calculated. Thus the 2x1- and 2x2-matrices result in

$$\underline{C}^*(\underline{x}) = \begin{bmatrix} N_A^{d_1} C_1\ (\underline{x}) \\ N_A^{d_2} C_2\ (\underline{x}) \end{bmatrix} = \begin{bmatrix} -\frac{S_h\ l_h}{\theta} \\ -\frac{c_5}{m} x_4^2 \end{bmatrix} \tag{15}$$

and

$$\underline{D}^*(\underline{x}) = \begin{bmatrix} \frac{\partial}{\partial \underline{x}}\ N_A^{d_1 - 1} C_1\ (\underline{x}) \\ \frac{\partial}{\partial \underline{x}}\ N_A^{d_2 - 1} C_2\ (\underline{x}) \end{bmatrix} \underline{B}(\underline{x}) = \begin{bmatrix} \frac{l_v}{\theta} & 0 \\ 0 & \frac{1}{m} \end{bmatrix} \tag{16}$$

respectively. These matrices are basic requirements to derive the nonlinear control algorithms which satisfy the universal equation

$$\underline{u}(t) = \underline{D}^*(\underline{x})^{-1}\Big\{ -\underline{C}^*(\underline{x}) + \underline{\Lambda}\ \underline{w}(t) - \underline{M}^*(\underline{x}) \Big\}, \tag{17}$$

where the control signals $\underline{u}(t)$ are generated in dependence from the state variables $\underline{x}(t)$ and the nominal values $\underline{w}(t)$. As the elements of $\underline{w}(t)$ correspond to the elements of $\underline{y}(t)$, the nominal values for yaw angle and velocity are determined by $w_1(t)$ and $w_2(t)$ respectively. By the 2x2-matrix $\underline{\Lambda}$, composed only by elements on the main diagonal, the nominal values are amplified, while by the 2x1-matrix $\underline{M}^*(\underline{x})$ pole assignment for the overall systems is carried out.

The existence of the inverse of $\underline{D}^*(\underline{x})$ is a necessary and sufficient condition for the decoupling of the system of differential equations (1). With respect to (16) the determinant of $\underline{D}^*(\underline{x})$ results in

$$\det\Big(\underline{D}^*(\underline{x})\Big) \;=\; \frac{l_v}{\theta\, m}, \tag{18}$$

whereby the inverse is calculated to

$$\underline{D}^*(\underline{x})^{-1} \;=\; \begin{bmatrix} \dfrac{\theta}{l_v} & 0 \\ 0 & m \end{bmatrix}. \tag{19}$$

To complete the control laws the matrices $\underline{\Lambda}$ and $\underline{M}^*(\underline{x})$ have to be specified explicitly. By

$$\underline{\Lambda} \;=\; \mathrm{diag}\big\{\lambda_1, \lambda_2\big\} \;=\; \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{20}$$

the nominal values are weighted with respect to the amplifications λ_1 and λ_2, while by

$$\underline{M}^*(\underline{x}) \;=\; \begin{bmatrix} \sum\limits_{k=0}^{d_1-1} \alpha_{k1}\, N_A^k C_1(\underline{x}) \\ \sum\limits_{k=0}^{d_2-1} \alpha_{k2}\, N_A^k C_2(\underline{x}) \end{bmatrix} \;=\; \begin{bmatrix} \alpha_{01}\, x_2 + \alpha_{11}\, x_3 \\ \alpha_{02}\, x_4 \end{bmatrix} \tag{21}$$

the dynamical behavior of the decoupled subsystems can be adjusted by the constant parameters α_{01}, α_{02} and α_{11}.

With respect to the universal formulation (17) and the corresponding matrices in (19), (20) and (21) the control laws are evaluated to

$$\begin{aligned} u_1(t) \;&=\; \frac{l_h}{l_v}\, S_h(\underline{x}) \;+\frac{\theta}{l_v}\,(\lambda_1\, w_1(t) - \alpha_{01}\, x_2 - \alpha_{11}\, x_3) \\ u_2(t) \;&=\; \quad T(\underline{x}) \;+m\;(\lambda_2\, w_2(t) - \alpha_{02}\, x_4)\,. \end{aligned} \tag{22}$$

As the control algorithms require the actual values for $S_h(\underline{x})$ and $T(\underline{x})$ of rear side force and air resistance, these variables are provided with respect to equations (4) in advance.

3.2 Dynamical behavior of the overall system:

With respect to the control algorithms (22) the dynamical behavior of the overall system is examined. Considering (1) and (13) these derivations yield

$$\begin{aligned} \dot{y}_1 \;&=\; \dot{x}_3 \\ \ddot{y}_2 \;&=\; \dot{x}_4. \end{aligned} \tag{23}$$

In this context d_1 and d_2 indicate, how often the output variables y_1 and y_2 have to be derived respectively. Replacing the first derivations $\dot{x}_3$ and $\dot{x}_4$ by the corresponding right sides of (1) and by further substitution of the control variables $u_1(t)$ and $u_2(t)$ by the right sides of (22) the overall behavior of the controlled vehicle results in

$$\begin{aligned} \ddot{y}_1 + \alpha_{11}\, \dot{y}_1 + \alpha_{01}\, y_1 \;&=\; \lambda_1\, w_1(t) \\ \dot{y}_2 + \alpha_{02}\, y_2 \;&=\; \lambda_2\, w_2(t). \end{aligned} \tag{24}$$

By these equations the overall dynamics of the vehicle describing in transverse direction a behavior of second order and describing in longitudinal direction a behavior of first order is determined. Desired variables are $w_1(t)$ for the nominal yaw angle and $w_2(t)$ for the desired velocity of the car.

With the free choosable parameters $\lambda_1, \lambda_2, \alpha_{01}, \alpha_{11}$ and α_{02} the dynamics of the controlled vehicle can be adjusted. It is sensible to introduce the following restrictions:

$$\begin{aligned} \alpha_{01} \;&=\; \lambda_1 \\ \alpha_{11} \;&=\; 2\,\sqrt{\lambda_1} \\ \alpha_{02} \;&=\; \lambda_2. \end{aligned} \tag{25}$$

In this way considering the stationary case the nominal values and the corresponding output variables are weighted by the same parameters. Furthermore the aperiodic borderline case is adjusted in transverse direction.

With the remaining degrees of freedom consisting of the parameters λ_1 and λ_2 rapidity of control circuits can be influenced by pole assignment. Here technical constraints of the vehicle have to be taken into consideration.

3.3 Modification of the nonlinear control laws:

As the value of the yaw angle, represented by variable x_2 in (22), is not measured and also a nominal value $w_1(t)$ for the yaw angle does not exist explicitly, an adequate modification of the relevant control algorithm is necessary. Rather the task is followed to find a control system, which adjusts automatically the car's direction of movement considering the course of the road. In this context the edge of the street is realized by the optical sensor mounted on the car. Orientations of the car and of the sensor are identical. So a control method must be found to adjust the yaw angle in dependence of the signal coming from the optical system. Due to these requirements in the control law for the yaw angle, which is represented by the first equation of (22), the variables x_2 and $w_1(t)$ have to be eliminated. In this way first the parameters λ_1 and α_{01} are equated to

$$\lambda_1 \;=\; \alpha_{01}, \tag{26}$$

so that the control error $w_1(t) - x_2$ can be factored out. The task is to substitute the control error in an adequate way. The deviation between the optical axis of the sensor and the edge of the street is determined by

$$\phi \;=\; \arctan\frac{l_{tr}}{l_{lo}}. \tag{27}$$

As the control system has to direct the vehicle on the right lane with nominal distance w_{ltr} to the right edge of the road, an offset

$$w_\phi(t) \;=\; \arctan\frac{w_{ltr}(t)}{l_{lo}} \tag{28}$$

has to be taken into account. By equation

$$w_1(t) - x_2 \;=\; w_\phi(t) - \phi \tag{29}$$

the control error is substituted by the difference $w_\phi(t) - \phi$. This relation is illustrated by Fig. 3. If further the parameters λ_2 and α_{02} are equated the control laws (22) considering (29) are transformed into

$$\begin{aligned} u_1(t) &= \frac{l_h}{l_v}\, S_h(\underline{x}) + \frac{\theta}{l_v}\Big(\lambda_1\ (w_\phi(t) - \phi) - \alpha_{11}\, x_3\Big) \\ u_2(t) &= \quad T(\underline{x}) + m\ \Big(\lambda_2\ (w_2(t) - x_4)\Big). \end{aligned} \tag{30}$$

Herewith based on the present kinematic structure and the available sensors transverse and longitudinal dynamics are controlled. The desired values are represented by the nominal distance of the path of the vehicle to the right edge of the street and by the nominal velocity.

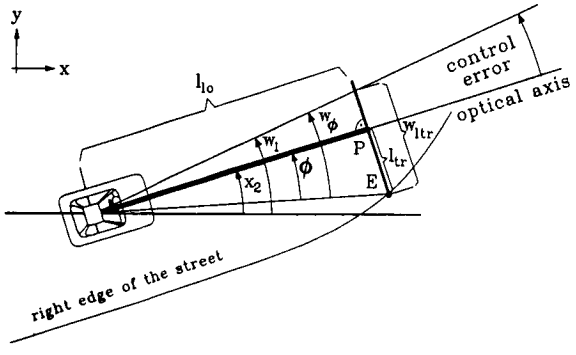


Figure 3: Determination of the transverse control error

3.4 Processing of original control signals:

By equations (30) the required values of the control variables for front side force S_v and rear driving or braking force H are computed. As the course of a vehicle is determined by the steering system, an algorithm is required to generate subsequently the corresponding value for the steering angle δ_v representing an original control signal in dependence of front side force S_v. In contrast control signal H for driving or braking force is correlated directly to the vehicle and thus it exists already as original control signal. For better distinction the original control signals $\bar{u}_1$ and $\bar{u}_2$ are marked by a bar.

Based on equation (3) and with respect to (2) the original control signal $\bar{u}_1$ is generated in dependence of the value S_v for front side force, which is represented by u_1 and evaluated by control law (30). In contrast the original control signal $\bar{u}_2$, which is identical to the desired value H for the rear driving or braking force, is represented by control variable u_2:

$$\begin{aligned} \bar{u}_1 \;&=\; \delta_v \;=\; \frac{u_1}{c_v} - x_1 + l_v\frac{x_3}{x_4} \\ \bar{u}_2 \;&=\; H \;=\; u_2. \end{aligned} \tag{31}$$

In this way the original control signals $\underline{\bar{u}}(t)$ are generated in a tandem arranged unit in dependence of vector $\underline{u}(t)$ and state variables $\underline{x}$ and subsequently they are transferred to the actuators of the vehicle.

3.5 Simulation results:

To demonstrate the quality of the control system, the dynamical behavior of the controlled vehicle was examined by computer simulations. Vicarious in Fig. 4 the resulting manœvre based on a sudden 45°-bend of the right edge of the street, which represents a hard test for the control system, is shown. This bend can be recognized by the graph marked with 'E', which consists of the fictitious path of the point E (see also Figs. 1 and 3). In contrast by the graph marked with 'P' the in the same way fictitious path of the point P, which is located on the optical axis of the sensor system and which is here of distance $l_{lo} = 5$ m to the car's center of gravity, is shown. It can be clearly recognized, how the vehicle, which drives a speed of 50 km/h, turns in this way that the controlled condition to keep the nominal distance of $w_{ltr} = 2$ m between the points P and E is followed. With respect to (25) the path of the vehicle's center of gravity drives through the bend with an aperiodic behavior and follows the street in

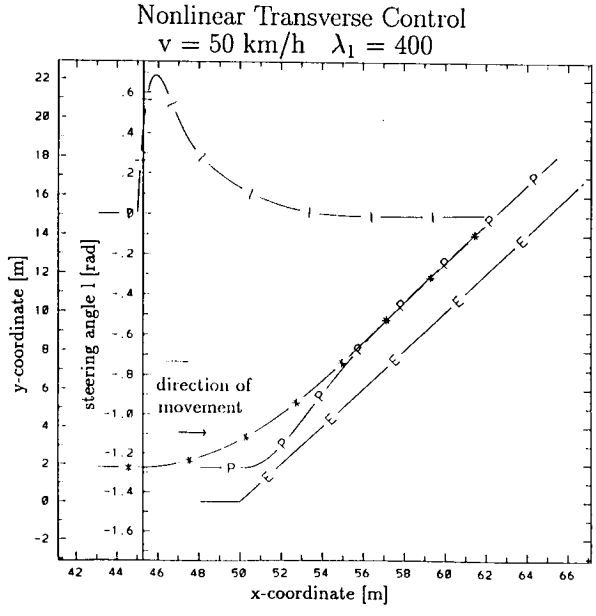


Figure 4: Dynamical behavior of the controlled vehicle

an exact way. Furthermore the course of the original control signal δ_v for the steering angle is described by the graph marked with 'l' in dependence of the actual position of the car on the x-coordinate.

4 Conclusion

An on-board system for transverse and longitudinal control of an automatically guided vehicle based on nonlinear decoupling strategies is presented. With respect to the transverse dynamics of the car the control signal for the steering system is generated in dependence of the information recorded by an optical sensor, which detects the edge of the road. In contrast the drive and the brakes are influenced by the longitudinal controller taking the nominal and the actual speed into consideration. Control signals for the steering system as well as for the drive and and the brakes of the vehicle are generated by nonlinear algorithms. This method is especially distinguished by its efficiency and its preknowledgement about the dynamical behavior of the overall system. Herewith an important requirement to increase safety in future traffic is realized.

References

[1] Freund, E.; Judaschke, U.: "Automated Vehicle Guidance as a Component of a Traffic Control System". Proceedings of the 2nd Prometheus Workshop, Stockholm, Sweden, October 1989, pp. 204–213.

[2] Y. Hossam El-Deen, Ali Seireg: "Mechatronics for Cars: Integrating Machines and Electronics to Prevent Skidding on Icy Roads". ASME, Computers in Mechanical Engineering, Springer–Verlag New York, Inc., Jan. 1987, pp. 10–22.

[3] Freund, E.; Lammen, B.: "Sensor Based Online Collision Avoidance for Autonomous Vehicles". Proceedings of the Second International Conference on Automation, Robotics and Computer Vision (ICARCV '92), Singapore, 1992.

[4] Freund, E.; Mayr, R.: "Path Control for Automatically Guided Vehicles". Proceedings of the 2nd Prometheus Workshop, Stockholm, Sweden, October 1989, pp. 224–233.

[5] Riekert, P.; Schunck, T. E.: "Zur Fahrmechanik des gummibereiften Kraftfahrzeugs". Ingenieur–Archiv, Vol. XI, pp. 210–224, 1940.

[6] Freund, E.; Syrbe, M.: "Control of Industrial Robots by Means of Microprocessors". Lecture Notes in Control and Information Sciences, Springer-Verlag, Berlin, Heidelberg, New York, 1977, pp. 167–185.

[7] Freund, E.: "Fast Nonlinear Control with Arbitrary Pole–Placement for Industrial Robots and Manipulators". International Journal of Robotics Research, Vol. 1, No. 1, 1982, pp. 65–78.

AIAA-94-1262-CP

THE PROBLEM WITH MULTIPLE ROBOTS

Marcus J. Huber and Patrick G. Kenny
Artificial Intelligence Laboratory
The University of Michigan
Ann Arbor, Michigan 48109-2110
marcush@engin.umich.edu, pkenny@eecs.umich.edu

Abstract

Research in multiple, robotic agents is gaining the interest of an ever increasing number of researchers. Many of these researchers have previously worked in simulation or with single robots, or both. Making the transition from a simulator to the real world can be very trying and frustrating to someone with no experience with such a project. The same goes for making the transition from a single robot to multiple robots. There are a number of issues that arise, mostly of the practical and pragmatic variety, that escape consideration by researchers making these transitions for the first time. We hope to highlight the most important of these issues – discovered primarily through experience with working on multi-robot projects, two of which are discussed in the paper – so that other researchers can give them full consideration when working on their own projects. In addition, we give some suggestions as to how to eliminate or minimize the negative impact these issues might have upon the development of a multiple robot project.

Introduction

A time will come when it will be common to see autonomous robots working together in teams or interacting as individuals. Each of these robots will be performing its specific role in achieving whatever it has been given as tasks. Individual robots, regardless of whether it is working in a team or not, will dynamically interact with each other, the environment, and with humans. They will communicate necessary information in noisy environments, fill in for fallen comrades, and adapt to the temporary loss of sensor subsystems. This scenario is still a long way in the future. What will it take to make this a reality? While research in Robotics and in Distributed AI is always pushing toward this future, research is still in its infancy compared to what is necessary before robots can function as described above.

Quite a bit of research has been done regarding issues related to multiple agents,[6,7,9] and some has been done specifically for multiple robots.[1,11] However, none of this work has really looked at what a researcher faces when trying to implement his or her ideas on real robots for the first time. In this paper we present a number of issues that arise when making the transfer from simulated, theoretical, or single-robot research to working with *more than one* real, autonomous, robot situated in a real world. We discuss each of these issues in some detail and give suggestions, based on experience, for dealing with them. In the first two sections we discuss the issues related to working with more than a single robot and those inherent to working with robots situated in the real world, respectively. We then describe concrete examples of the problems faced when working on multibot projects. Throughout the paper we give suggestions on what can be done to eliminate or reduce these kinds of problems.

Multibot Issues

A number of issues arise when working with multiple robots. We divide these roughly into the issues that arise when looking at the collection of robots as a whole, and those issues that arise when looking at the individual robots that make up the collection. Many of the "collective" issues (such as those that deal with communication, organization, cooperation strategies, etc.) have been addressed in Distributed AI (DAI) research and tend to be fairly abstract in their nature. We talk briefly about these issues but we do give pointers to where more in-depth discussions can be found. Issues that arise from the collection of "individuals", primarily due to the heterogeneity between the agents, seems to be a topic of research of interest to a great number of fields of study (robotics, DAI, artificial life, etc.) but to no one field in particular. In this section we discuss the issues that we see are the most significant to researchers working with collections of these physically embodied agents (robots).

This research was sponsored by DARPA under contract DAAE-07-92-C-R012.

Copyright © 1994 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

Heterogeneity

Heterogeneous het.er.o.ge.ne.ous , *adj.* Consisting of or involving parts that are unlike or without interrelation; having dissimilar elements; not homogeneous. [< GETERO- + Gk *genos*, kind, sex.] -het'er*o*ge*ne'i*ty *n.*

Heterogeneity among a collection, group, or team of robots is a **BIG** issue. In fact, it may be the single most important issue for researchers making the transition from simulation or theoretical work to consider. Research performed in simulation seldom lacks the completeness required to fully model the differences between robot platforms that will serve as the real-world implementation. Very small differences between simulated robots, which appear insignificant to the uninitiated, can become overwhelming when real robots are pressed into service. We identify a number of factors to especially watch out for (with respect to differences in the robotic platforms) in order to make the transition to real robots easier, and to reduce the potential impact upon various aspects of the multiple robot system if the differences are not eliminated or reduced.

Robots come in an incredible variety of sizes, shapes, and capabilities. Robots can be arms, mobile bases, gantries, snakes, or any of a number of other alternatives. This richness of design makes for a wide range of applicability of robots to different domains, environments, and applications. It is also a major source of grief for anyone wishing to do research with more than one of these robots. There are a great number of places where heterogeneity can cause problems. We divide these into innate and non-innate characteristics, discussed below.

Innate

We consider innate characteristics of a robot to be those features that a robot is "born" possessing, those which are inherent to a robot's basic design and is generally determined by the manufacturer. These include: physical characteristics such as weight, size, and shape; precision and/or accuracy of such things as odometry, positioning (e.g. robot body, camera, etc.); modality and number of sensors; characteristics of the low-level control such as dynamics, functionality, interfacing; design limitations and characteristics such as holonomic characterization, the number of degrees of freedom, bounds on speed, acceleration, reach, etc., and carrying capacity; and the number and type of actuators and/or manipulators.

Many of these features are either impossible, or very difficult, to change, remove, or replace, and are a major boon and bane of robotics researchers. A robot that comes with a powerful, flexible, and complete set of innate "features" can be greatly advantageous. And conversely, a poorly designed robot, or one that may be designed well but ill suited to the task to which is applied, can be a nightmare.

Non-innate

We call the features and characteristics of a robot which are in the control of the roboticist the non-innate features of the robot, those which are a result of work done on the robot to add to or change the functionality of the robot after it arrives from the manufacturer. This includes such things as: the number, modality, precision, etc. of sensors; the number and type of actuators and manipulators; the number, power, memory, connectivity, etc. of processors; the programming language; the high-level control scheme (if any, which would then include the high-level control interface to the low-level controller); the inter-agent communications modality and characteristics; and "sugar" features like speech synthesis and recognition capabilities, graphics displays, etc. It might also include those innate characteristics that can be modified, as there may be some fuzziness to the distinction. There is generally a greater variation in the non-innate features of a robot than in the innate features, due to the wide range of add-on and upgrade possibilities, including "homemade" designs.

The problems

Heterogeneity is an inherently multi-agent issue, as it is defined as the existence of differences between two objects, in this case robots. Heterogeneity arises from both the innate and non-innate features of the robots, and may be looked upon as an advantage in situations where the heterogeneity can be exploited. However, the differences between robots can, and usually does, eventually cause problems. The problems associated with innate and non-innate feature can be very similar, but may possibly have very different solutions (as discussed below). As mentioned earlier, heterogeneity between the robots might very well be *the* most important issue to be faced by researchers working with multiple robots. Our empirical intuition is that the difficulty of implementing and maintaining a collection of multiple robots is a function of both the heterogeneity and the number of robots. We believe that the relationship is something like that of Figure 1. As you can see, we believe that the difficulties associated with increased numbers of agents increases at a higher than linear rate. We believe the same follows for heterogeneity. Of course this is totally unsubstantiated, and is based solely on past experience with multibot implementations.

The problems caused by heterogeneity usually manifest themselves not in the actual experiments conducted by the researcher, but in the *development* stage of the research, where the robots are being readied for the experiments. The development period usually serves the purpose of dealing with the differences, either to avoid them or to take advantage of them, so that when the robots are ready to run experiments the issues have already been considered and addressed. While designing and implementing the robots' sensors, control systems, processing hardware, coordination scheme, etc., a researcher may face problems in any of a number of areas, which we have divided into three broad categories: software, hardware, and functionality. For each category we describe the source of problems that can occur and their effect upon the development of a multibot system.

- Software - Software on the various robots in the

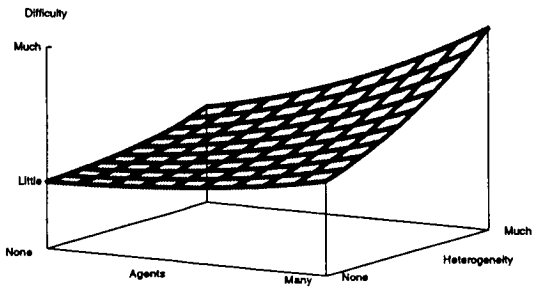


Figure 1: The relationship of difficulty of implementation and maintenance to heterogeneity and the number of robots.

"collection" may be affected by differences between any of a number of robotic characteristics, including the processors, compilers, programming languages, sensors, speed, development environments, and third-party software libraries of the various robots. Any difference in these, or any other of the innate or non-innate features, may create the necessity to modify software to suit a particular robot, which will make the robot all the more heterogeneous. Research agendas themselves may force differences in the software systems utilized by different robots, such as requiring different control architectures or obstacle avoidance algorithms, in order to study the tradeoffs associated with them. Differences in software may range from changed parameters, to modified code, to different software modules, to completely different software systems. Regardless of the source and extent of the heterogeneity, once the differences occur it can be a nightmare to make changes across all of the involved robots to account for each robot's idiosyncrasies.

- Hardware - Robot hardware may differ in sensors, mechanics, physical dimensions, dynamics, CPU's, equipment storage volume, etc. This may be a result of having purchased the robots at different times, implementing different sensor system designs, replacement of broken equipment with non-original parts, etc. Robots that are dissimilar in hardware may or may not create problems; If the hardware on different robots is not equivalent, in that there are enough differences in functionality, modality, speed, etc. to not be transparently switchable, software problems like those discussed above will most likely be created. And other difficulties may also arise, such as having to gain expertise on more and more varied equipment and maintaining the various robots' different hardware.
- Functionality - The capabilities that a robot has depends upon the combination of hardware and software that it has. Given a robot with a particular hardware configuration, the robot can have a range of functionality, depending upon the software written to use the hardware. Likewise, given control software and sensing algorithms, the robot can have varied functionality dependent upon the characteristics of the actual sensors, manipulators, drive motors, and other hardware that the robot is fitted with. Heterogeneity in any aspect of a robot, be it sensing, control, motion, manipulation, or some other aspect, creates a situation where the researcher must make a decision about the functionality that he/she wants the robots to actually possess. Emphasis may be on having all robots possess the same functionality, or it might be desired that the robots possess the maximal functionality possible. Choosing the latter, while understandable, causes more heterogeneity than the former*, and hence possibly exacerbates future problems similar those items discussed above.

Suggestions

The single most important suggestion that we can make to researchers is that they reduce the amount of heterogeneity in the robots that they work with. Heterogeneity between robots is probably the single largest source of problems, effort, and grief encountered while working on research. Eliminating all differences between robots would be ideal, of course, but is not always possible. Robots from different manufacturers will certainly have differences in innate characteristics, as will different models of robots from the same manufacturer, as will even the same model robot from different years. However, these differences can be eliminated *at some level of abstraction*, and it is our suggestion that an effort should be made to accomplish this.

For example, if two robots differ in their low-level motion control functions, a set of higher level functions can be built on top of these commands that removes the robot-dependent aspects. Code written using this new set of functions can then be readily ported between robots.

Of course, dealing with heterogeneity is an interesting research topic, and is therefore necessary in some situations. But it is our belief that it is much easier to introduce differences in robots by disabling functionality or changing parameters (as examples) than it is to eliminate or reduce differences.

Communication

When we talk about communication among robots we mean the intentional act of trying to convey information. And, while communication may not explicitly be used by some researchers,[1,11,13,15] it is very common.[5,8,12,14] Communication between robots is most commonly accomplished using some form of radio frequency (RF) transmission, although it might

*Unless all the robots are exactly the same in all respects, so that their maximal functionality is exactly identical and all the software and hardware required to reach this functionality can also be identical. If the robots are *not* exactly the same, the heterogeneity will show up in the software, at least, in order to achieve the same functionality, if this is even possible.

eventually become possible to explicitly pass meaningful amounts of information by visual means. Tethers or other physical links will most likely not work except for robots firmly fixed in place, such as robotic arms that are not on mobile bases.

Communication can be accomplished in a number of ways, including simple point-to-point and broadcasting (ie. to all robots within range). Complex multibot communication networks can be constructed, however, where robots may not only act as recipients and originators of messages, but also as relays, helping pass messages between two other robots. As more robots interact, communication issues become more and more of an important issue.

Communication issues are unique to multiple robot scenarios (if only because it generally does not make too much sense for a robot to send messages to itself); problems related to communications are therefore also unique to multiple robots.[†] Through our endeavors in multi-robot research, we have identified a number of the problems that seem to plague communication, and we have identified some practical suggestions to at least reduce these problems. Some of the more significant problems are listed below:

- Missing messages - messages never get to their destination.
- Wrong messages - the wrong message is sent, an agent intercepts a message meant for another agent and mistakenly takes it to be for itself, or a message header gets corrupted in transmission and is sent to the wrong agent.
- Garbage contents - a messages information is corrupted to the point of uselessness.
- Communications hardware failure - an agent suffers total loss of ability to communicate.
- Transmission delays - A message's arrival is delayed due to length of travel, number of relaying robots, etc.

All of these problems are caused by RF noise either corrupting or overpowering the intended communications. The magnitude of the problems one will face is directly related to how noisy the RF environment is in which the robots will be used, how robust the low-level communications hardware and software is to corruption (via error checking and correction, handshaking, etc.), and how robust the abstract coordination mechanism is that the robots are using in order to work together (higher level protocols, negotiation schemes, etc., if used at all).

Some more abstract issues related to communication, many of which have been studied in Distributed AI literate, include common knowledge, synchronization, and coherence. Working with real robots means that there is *always* a chance that a message will not be received by the intended robot, or that if it is, that it is corrupted. Halpern and Moses,[10] prove that the involved agents cannot be sure of achieving common knowledge about *anything* that requires communication in such situations. Synchronization of agents is related to this in that, quite often, communication is used by the agents to reach a common point in time at which they know each other's "state" (and can then go on to perform coordinated activities, guaranteed non-interfering actions, etc.)[16,17] Synchronization is usually only possible, however, when common knowledge of every involved agents' state exists so that they can realize when synchronization has been achieved. Coherence deals with coordinating agents having compatible and non-contradictory information. Coherence can be achieved through communication of the data itself, supporting or conflicting evidence, etc. so that each agent eventually believes compatible information.

Of course, if the interacting robots are unconcerned with explicitly coordinating with other agents they will most likely not communicate (as in[1]), and therefore not reason about these communications-related issues.

Suggestions

Solutions to deal with communication problems are pretty commonplace. Technical solutions for these problems include retransmission of messages, semantic message content checks, acknowledged messages, periodic confirmation of activity ("I'm alive!") messages, addressed messages, and robust error detection and correction protocols, among others. Different techniques are necessary for variations of domain, application, robot organization, environment, etc. For instance, in extremely noisy environments it might be necessary to employ error detection and correction mechanisms, retransmission of messages, and handshaking protocols. When the robots are prone to failure, but the environment is noise-free, using simple communication protocols might suffice, but periodic messages from agents indicating that they are functioning might be useful. In general, design in communication overkill. Buy high power, flexible, high quality communication hardware. Determine what will be the worst possible environmental noise that the robots will face, and then employ techniques discussed above for environments twice as noisy.

Planning, Organization, and Task decomposition

Issues related to planning, organization, and task decomposition, among other abstract multi-agent issues, are beyond the scope of this paper, and much research has been conducted on these topics in the Distributed AI field. See "Readings In Distributed Artificial Intelligence"[2] for a collection of papers dealing in detail with these issues. What should be mentioned here, however, is that each of these issues may, in some manner, be affected by the issues discussed in the rest of this paper. Some examples are given below:

- Heterogeneity - Multiagent planning routines will have to be modified to deal with heterogeneous

[†] Communication from a single robot to a base station is pretty common these days, so those researchers that do this will have some insight into multi-robot communication problems.

robots in order to model each of the robots' individual characteristics. As the robot's might differ in any of a number of ways, this might mean adding a large amount of complexity to the planner. Organization(s) of robots will be less flexible than for homogeneous robots, as each agent may not be able to fulfill the responsibilities of every other agent in the organization. Much like the planning system, task decomposition might be much more complex in order to account for the differences in the robots.

- Communication - Communication issues can be expected to work with the equipment and protocols that are going to be used, what type of task decomposition makes the most sense given the environment and domain

Real World and Simulation Issues

Working with robots in the real world may, at first glance, seem to be an easy extension of similar research performed in simulation. And, while simulators are good for testing theories, debugging designs, or just running tests due to environmental constraints, a simulator may give the false impression that the real world is simple and predictable. However, the world is not exact, and is much more complex then any simulator can realistically model. However, they should not be considered satisfactory models in which to completely design and develop algorithms and paradigms destined for real–world robotic applications. Toward this end, some researchers have totally foregone the use of simulators, and have opted to use the world as its own model.[3] We should only rely on simulators to help us prepare a robot for the real world. Though we agree that simulators can be a valuable tool to supplement research with physical robots,[18] at some point in the development cycle it will become necessary to make the transition to the real world. This may not be such an easy task for there are many issues that need to be considered. We divide these issues into three categories. These are: issues that deal with the robot hardware and platform, issues that deal with robot sensors and actuators, and issues that deal with robot software.

Hardware

The first issue, robot hardware and platform, is arguably the area with the most substantial differences between simulation and the real world. Most simulators do not simulate real world events such as when the battery gets low, when computers and sensors on a physical robot fail or start to misbehave (usually without one being aware of the fact), when motors degrade or burn out, or when gears or wheels slip or break. There is a great amount of hardware, all of which, usually, has to work flawlessly. There is hardware to control the motors, hardware to control the sensors, and usually a central computer. There is also minor hardware appliances such as batteries, power converters, actuator circuitry, and monitors that may need to be on board.

An issue that may be overlooked quite often is the actual space requirements of sensors and other hardware on a robot. While a simulator may simulate the functionality of each of the physical components, it probably does not simulate how to place all these components onto a robot base while keeping it stable, usable, and accessible. Robots in a simulator will have an array of simulated sensors, perhaps covering a gamut of modalities such as sonar, vision, range imaging, structured light, and infrared. The robots may have to communicate with each other. They may have manipulators of various sorts. A simulator may permit a all of these capabilities on a robot without consideration to the practicality of doing so.

The type of robot platform or base that the robot is built on will is another important issue to consider. If a base is purchased from a manufacturer, one may not need to worry about the base design, but one does need to worry about the ability to add hardware and software to the robot (i.e. change the non-innate features). Consideration must also be given to such details as the base drive design, which can take a wide variety of forms, from differential drive or synchro–drive, to a tricycle–like or car–like design. The different drive systems may mandate some design decisions on the type and use of sensors, the type of robot control software used, the planning system, etc.

Power constraints are another serious issue often overlooked when dealing with real robots. A great deal of a robot's life is spent charging its batteries, the more so if it is heavily loaded with sensors, actuators, and other electrical equipment. Also, the size and number of batteries restricts the amount of power (current) available, strictly limiting the amount of electrical equipment that may run concurrently on the robot. Even when within this limit, the battery life of a robot is determined by the load on its batteries while running. A robot that must operate over a long period of time must therefore have a light complement of electrically driven sensors, actuators, and other electrical equipment.

Sensors

The second issue to consider are the sensors that a robot might use in the real world. One must consider the kinds of sensors that will be necessary for the robot to perform its task and how many sensors the robot is going to need. A simulator can only approximate the data returned from a sensor based upon its internal sensor model. The more accurate or complex the sensor , the more sophisticated the model will have to be. As an extreme point of view, it may not be possible to realistically simulate real sensors (except perhaps extremely simple sensors like limit switches) in a simulator due to the complexity and uncertainty of the real world. No one can can anticipate *all* of the possible ways that a sensor may fail and/or err. Problems that can (and do) occur include cameras that are not calibrated to specifications, lenses that are dirty or misaligned, sonars with shorted circuitry, incorrect sensor values due to low input voltage, and motor decoders that perform differently than specified or degrade in performance over time.

Another issue to consider is whether the sensors

modeled in simulation return realistic data. For example, a simulated robot may have a sensor that can return the identity of a nearby human. But, one must ask, is there a sensor in the real world that can do that? Perhaps, but probably only at the cost of a great deal of design and implementation effort, and probably one that is quite fallible.

Software

It may be quite a formidable task to track down a software error on a real robot. In a simulator, the only place an error would normally arise would be in the user developed code. In the real world, one also has to consider that an error that might cause the robot to fail or perform differently from the simulator may be the result of a sensor failure, low battery, computer failure, cable problem, or some other seemingly unrelated cause.

Depending upon how carefully the software was designed with the real world in mind, the software run in simulation will most likely run in the real world. When tested in the real world, however, one may find that algorithms need to be changed, libraries modified or rewritten, parameters changed, etc. When making the transition to real robots, one must also consider the size of the software and speed of the robot's processor. Porting code to a minirobot will require great creativity if the software was developed on a Cray supercomputer and requires 128 megabytes of memory. Due to the increased complexity of the real world with respect to that in a simulated model, robot tasks will always be harder than that faced in a simulator. It may require more subtlety or complexity in the planning and control software, more sensors and computational resources, or integration of software not included in the simulator model, to accomplish even a simple task.

Suggestions

To ease the transition of a robot design from a simulator to the real world we offer several suggestions.

- Expect that some additional code will be needed on the physical robot. Some helpful test programs that will test each component to assure that it is working properly, whether it is a sensor or a subroutine, will be useful. After three weeks of running a camera, for example, it may have been bumped out of alignment or had its aperture closed so that it is causing strange and unexpected results in the robots behavior. Hardware problems can have very misleading symptoms and it is good to have test programs that can easily rule out simple causes. And always check connectors, as they are apt to become disconnected. This is a very common problem, as everything on the robot uses cables (cameras, disk drives, monitors, keyboards, etc.) and simply the vibration caused by the robot moving around is enough to loosen cables.
- A useful place to look when a problem arises is the battery. A low battery may cause errors that did not previously exist and be of a type that has never been encountered before. A robot may work correctly for months and then develop unexpected errors because the battery is running low.
- It always helps to work in stages. Test each component and routine as it is moved from the simulator to the real robot(s).
- Document the errors that are encountered. One does not want the next person working on the robot(s) to repeat the same mistakes.
- Last of all, always have a remote emergency robot stop button handy, especially for large robots. Some robots can move very fast and weigh several hundred pounds, and may unexpectedly go out of control.

It may seem that the task of transitioning a robot design from a simulator to the real world will be quite simple and straightforward. There are many issues that need to be considered. There will always be unexpected problems to deal with, but a user that is prepared and considers the issues mentioned above might ease the transition.

Examples

We recently worked on a couple of projects that involved the cooperative interaction of two heterogeneous indoor mobile robots. One project, involving exploration and navigation of an office–like environment, was implemented with some care and consideration to the issues described in this paper, but was so beset by problems that it failed to be completed within a hard time deadline and was never fully implemented. Another project, where the robots had to push boxes across an obstacle strewn floor, was implemented extremely quickly and serves as an excellent example of where failure to implement according to the suggestions above resulted in a very brittle, failure–prone, and frustrating multibot system but which, with some effort to resolve the problems using the suggestions in this paper, can become much more robust and successful.

Dynamic Duo

One of the two robots involved in these projects is CARMEL, a mobile robot based upon a Cybermotion K2A mobile platform. BORIS, the other robot used in the projects, is based upon a TRC Labmate platform. These robots are shown in Figure 2. The major innate difference between the platforms lies primarily in the motion characteristics - CARMEL is essentially holonomic, being able to move in any direction at any time without first having to turn its body, while BORIS must first turn to face the direction of travel before moving. Hardware differences other than those of the platform itself consisted of substantially different sonar rings (CARMEL has a circular ring of sonars, BORIS only has sonars facing in the forward direction), different CPUs (not only do they have IBM 80486 compatibles with differing characteristics, but CARMEL had an IBM XT compatible running the sonar system that BORIS does not have), and differing vision systems (CARMEL has a camera mounted on an independently rotating table, while

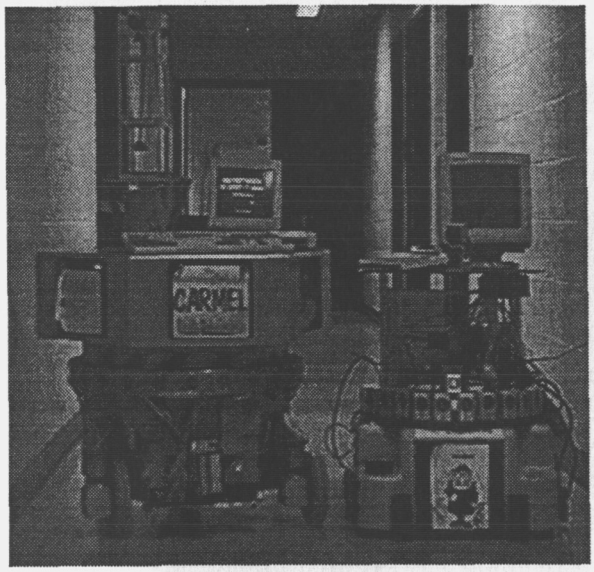


Figure 2: CARMEL (left) and BORIS (right), robots used in several multibot projects.

BORIS's camera is fixed to BORIS's top facing directly ahead of BORIS.) Low level functionality, was similar, but the robots had different implementations of some basic functions, such as obstacle avoidance.

Exploration

One of the events in the National Conference of Artificial Intelligence's '93 robot competition was to autonomously explore an "office" environment looking for a visually tagged object, which was then to be delivered to a predetermined room in the office complex. Walls in the "office" were three foot high panels of sonar–reflecting plastic and arranged to form rooms and halls. Each entry's robot was placed within the office without knowing where within the office it was or in which direction it faced. However, the robot(s) were told in which quadrant of the office environment they started. And each robot was given mostly accurate knowledge of the office layout with respect to wall placement and metric measurements; the actual office layout could differ from the map in that some doors could appear where not indicated on the map, and some doors on the map could disappear.

While most entries were single robot approaches, which we also tried with CARMEL, we attempted a multibot approach (both BORIS and CARMEL) as we saw a distinct advantage in exploring the office in parallel with two cooperating robots. However, while CARMEL was already a fully autonomous mobile robot, with obstacle avoidance and vision systems and a great deal of experience in research and robot competitions,[4] BORIS was initially only a bare TRC Labmate robot base upon which a small amount of obstacle avoidance research had been performed. We realized from the onset that we should attempt to design BORIS to be as *functionally* similar to CARMEL as possible.

A great number of engineering changes had to be made both in hardware and software to accommodate the many differences in the two robots. An example is the feature detection algorithms used to detect the various configurations of "office" halls and openings. The same basic sonar–based obstacle avoidance system was pre–existent in the robots before development started, so that including algorithms to perform feature detection seemed straightforward. The two robots had very different implementations of the sonar system, however, which made porting of the algorithms developed on one robot to the other robot extremely time consuming and is perhaps the single most important factor in the difficulty we had implementing the multibot approach. In addition, the sonar hardware differences (complete vs. partial rings of sonars) require BORIS to rotate in certain situations when CARMEL does not have to, requiring modifications to some low–level functions as well as the planner. Parameters in the sonar system also had to be fine–tuned to each robot to account for differences in the robot's size, sonar filtering system implementation, and other factors, and we were never able to perfectly match the results of the two robots.[‡]

Box pushing

The box pushing task at the AAAI '93 robot competition involved locating boxes (marked with distinctive visual tags) and moving them into a predefined pattern while avoiding obstacles (boxes that were not to be moved). Because a robot pushing a box cannot "see" with its sonars and therefore cannot perform obstacle avoidance, this meant that one robot either scout out a clear path beforehand and then go back and get the box, or that two robots help each other, with one robot acting as the navigator and one robot pushing the boxes. We chose to implement the multibot design, which made the design more challenging, difficult and, as we found out, frustrating.

Our approach to the task was to use CARMEL as the "Boss", or navigator, and BORIS as the "Worker", or box pusher. This task assignment was made because BORIS is built upon a square base, making it more conducive to pushing boxes than CARMEL, which has a cylindrical base. It was important that each robot have some shared knowledge of the world to properly navigate around the arena; for CARMEL to act as the navigator and tell BORIS where to push boxes, it was necessary that each robot initially knew where the other one was located in a global coordinate system. Each robot also had an internal map of the arena indicating the boundaries and the single interior wall. Obstacle and object locations were *not* known beforehand.

The interaction of the robots in this task was defined as follows:

The robots first job was to synchronize themselves using a sequence of handshaking messages. Once synchronized, BORIS would drive to one of a number

[‡]Further ramifications of the difficulty experienced with implementing the multibot approach was that development of the single robot technology was slowed a great deal due to the "thinned" person–power of trying to bring two robots up to competition speed instead of only one.

of predetermined viewing positions in the arena and look for boxes. If it found one (or more) it would approach it, stop just before the box, and communicate its location and orientation to CARMEL. Meanwhile, CARMEL would be waiting at its initial location until it received this message. CARMEL would travel to a location in line with the goal point where the box was to be dropped off and a few meters in front of BORIS's position. CARMEL would then travel in three meter segments to the goal point while avoiding obstacles. At each of these via–points, CARMEL would send its current position to BORIS as an obstacle–safe position. BORIS would attempt to move between these points in such a way as to keep the box securely in front of it (using relatively slow, wide turns) and would "follow the leader" to the point that the box would be placed. Because CARMEL moved in such small increments, the hope was that BORIS's path between via–points would keep it away from obstacles. Once CARMEL reached the box dropoff location, it would then move a safe distance away and wait for BORIS to find another box. BORIS would continue to push the box until the final location was reached. Free of the hindrance of the box, BORIS could then use its own obstacle avoidance system to move to the next viewing position and search for the next box.

As expected all did not work out the way it was planned. There were some problems encountered, some with BORIS, some with CARMEL, and some with the cooperative aspects of the task. BORIS's problems began with its vision system. On the initial attempt, the camera BORIS was using to locate boxes was pointed too low. As it turned out, the boxes we used for testing the robots were upside–down compared to the ones actually used in the arena, causing the tags on the boxes to be higher than the camera could see. BORIS moved from box to box, not recognizing anything, while CARMEL sat motionless waiting for a message from BORIS. After approximately five visual scans, BORIS suffered an unexplained lockup and we had to restart. We fixed the camera angle on BORIS and tried again.

On the second run, BORIS located the box correctly, approached it and transmitted a message to CARMEL communicating he was ready for the 'bosses' orders. CARMEL's map had been initialized improperly, so CARMEL thought that it was on the opposite side of the arena. It moved to the correct location in its own map where it thought BORIS was, not to where BORIS actually was, and plotted out a path to the goal position. The map error resulted in a large discrepancy in positions, however, and the resulting path was useless. Unaware of this, CARMEL then sent the command to BORIS telling it to proceed. CARMEL proceeded to move along its planned path toward the phantom goal destination, sending position messages that were uncorrelated with BORIS's map. BORIS, also unaware of the problem, started to execute the navigation path transmitted from CARMEL, but started to drive in the wrong direction and we had to start over again.

On the third and final run, the situation improved slightly. The robots synchronized, BORIS immediately found a box, CARMEL acknowledged the communications, moved in front of BORIS and the box, and traversed an obstacle–free path for BORIS to travel. CARMEL moved a safe distance away from the box drop off position and waited for BORIS to complete its pushing. However, at some point in the transmission of the initial path via–point from CARMEL to BORIS the message was corrupted and values for parameters were radically in error. This caused BORIS's control program to crash and hang, leaving both robots hanging.

While implementing the design for this project we ran into a number of issues discussed in this paper. Because we were able to actually implement and perform this task, we encountered both issues like those encountered during the exploration task as well as run–time and coordination issues that we did not have an opportunity to discover in the abortive attempt at the exploration task. Communication related issues figured quite prominently among those we ran into. Both during development and actual competition runs we experienced delays, losses, and corruption of messages sent between the robots. Because of the short time period in which we implemented our design (approximately 24 hours), we were unable to build in many of the safeguards recommended above. We successfully synchronized the robots using a set sequence of messages between the robots. We did not, for example, implement any form of semantic contents checking to detect invalid messages. Nor did we implement high–level retransmissions of messages if an expected response from the other robot never arrived. Heterogeneity in hardware did not surface as a significant issue for this task, primarily because we used the robot's heterogeneity to best advantage, we were not trying to achieve equivalent functionality, and the robots were to work have different task responsibilities. Software was more of an issue for the exact same reasons; we had to develop entirely different control software for each robot and could not develop software on one robot and port it to the other.

Future Work

The examples above give vivid accounts of the problems likely to be faced by researchers working with multiple real robots. We continue to be interested in multibot systems and applications, despite the extra effort that such work entails. The next major project is to develop a team of cooperating outdoor robotic vehicles (new military jeeps called HMWMMV's, or "Hummers") that can perform a task such as forward reconnaissance. We have designed and built one prototype vehicle (MAVERIC) based upon an electric utility cart, with which we can do development and experimentation without requiring access to the large and costly Hummers. When building another vehicle – to develop the multiple robot coordination technology necessary for this type of task – we will pay great attention to the issues raised in this paper. First and foremost, we will

try to make the two vehicles as identical as possible. This will most likely entail outfitting the second vehicle in exactly the same manner as MAVERIC in many aspects. It will also probably require upgrading MAVERIC with improvements, based upon our experience with MAVERIC and discovery of shortcoming in its design or implementation, that will be implemented from the onset on the new vehicle.

We would also like to explore using minimal resources on multiple minirobots to perform cooperative tasks. The robots that we are using are based on the MIT miniboard and use an MC6811 microcontroller. While computing power and controlling software are the same for each of the robots, there may be some small variations in the bases and sensors. The general idea is to use the ideas we have discussed here to help design robust multiple robots and apply them in the real world.

Conclusions

A great deal of research has been performed regarding how we can get multiple agents cooperating together in wondrous harmony. Much of this research has not, as yet, involved working with the agents that will actually end up doing the work, robots. A researcher who has not already attempted this technology transfer is in for a lot of headaches unless he or she has some insight to the issues that are associated with such a process. We have introduced and discussed the most significant issues and their causes, and have given many suggestions on how to deal successfully with them. A great deal of work and aggravation can be avoided by paying attention to these issues *before* implementation of a system on real robots. We have described the problems and solution faced when implementing two multibot projects in particular in order to fully illustrate what might occur during implementation, and accentuate the importance of paying heed to the issues raised in this paper.

References

[1] Ronald C. Arkin. Cooperation without communication: Multiagent schema-based robot navigation. *Journal of Robotic Systems*, 3(9):351–364, 1992.

[2] Alan H. Bond and Les Gasser. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[3] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, RA-2(1):14–22, March 1986.

[4] Clare Congdon, Marcus Huber, David Kortenkamp, Kurt Konolige, Karen Myers, Alessandro Saffiotti, and Enrique Ruspini. CARMEL vs. flakey: A comparison of two winners. *AI Magazine*, 14(1):49–57, Spring 1993.

[5] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. On the utility of multiagent autonomous robot systems. In *Working Notes: Workshop on Dynamically Interacting Robots*, pages 101–108, Chambery, France, August 1993. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[6] Edmund H. Durfee and Victor R. Lesser. Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 875–883, Milan, Italy, August 1987. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 285–293, Morgan Kaufmann, 1988.).

[7] Eithan Ephrati and Jeffrey S. Rosenschein. Constrained intelligent action: Planning under the influence of a master agent. In *Proceedings of the National Conference on Artificial Intelligence*, July 1992.

[8] Michael Georgeff. Communication and interaction in multi-agent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 125–129, Washington, D.C., August 1983. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 200–204, Morgan Kaufmann, 1988.).

[9] Michael Georgeff. A theory of action for multiagent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 121–125, Austin, Texas, August 1984. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 205–209, Morgan Kaufmann, 1988.).

[10] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In *Third ACM Conference on Principles of Distributed Computing*, 1984.

[11] Marcus J. Huber and Edmund H. Durfee. Plan recognition for real-world autonomous agents: Work in progress. In *Working Notes: Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots*, AAAI Fall Symposium, pages 68–75, Boston, MA, October 1992. American Association for Artificial Intelligence.

[12] Kouji Ishioka, Kazuo Hiraki, and Yuichiro Anzai. Cooperative map generation by heterogeneous autonomous mobile robots. In *Working Notes: Workshop on Dynamically Interacting Robots*, pages 58–67, Chambery, France, August 1993. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[13] Maja Mataric. Synthesizing group behaviors. In *Working Notes: Workshop on Dynamically Interacting Robots*, pages 1–10, Chambery, France, August 1993. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[14] Ei-Ichi Osawa. A scheme for agent collaboration in open multiagent environments. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 352–359,

Chambery, France, August 1993. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[15] Lynne Parker. Learning in cooperative robot teams. In *Working Notes: Workshop on Dynamically Interacting Robots*, pages 11–23, Chambery, France, August 1993. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[16] Jeffery S. Rosenschein. Synchronization of multi-agent plans. In *Proceedings of the National Conference on Artificial Intelligence*, pages 115–119, Pittsburgh, Pennsylvania, August 1982. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 187–191, Morgan Kaufmann, 1988.).

[17] Christopher J. Stuart. An implemenation of a multi-agent plan synchronizer. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1031–1033, Los Angeles, California, August 1985. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 216–219, Morgan Kaufmann, 1988.).

[18] Mark C. Torrance. The case for a realistic mobile robot simulator. In *Working Notes: Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots*, AAAI Fall Symposium Series, pages 181–184, Cambridge, Massachusetts, October 1992. American Association for Artificial Intelligence.

A GENERIC TELEROBOTICS ARCHITECTURE FOR C-5 INDUSTRIAL PROCESSES

Paul G. Backes†, Wayne Zimmerman†, and Michael B. Leahy Jr.‡

†Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

‡Air Force Material Command
Robotics and Automation Center of Excellence
San Antonio Air Logistics Center
Kelly Air Force Base, Texas

Abstract

The application of telerobotics to aircraft depot maintenance and remanufacturing is described and a telerobotics architecture for the application is discussed. Telerobotics will enhance process quality and could potentially decrease turn-around time and costs while moving human operators from hazardous work areas to safe and comfortable operator control stations. The approach is to augment, not replace, the human operator by blending human skills with robotics capabilities. Configurations of the architecture for telecrane, mobile carrier, and gantry applications are shown.

1. Introduction

This paper summarizes a study performed by the Jet Propulsion Laboratory for the Air Force Material Command (AFMC) Robotics and Automation Center of Excellence (RACE) to evaluate the feasibility of telerobotic solutions to C-5A heavy lifter aircraft maintenance processes and develop a telerobotics architecture for the application [1]. The architecture was developed for general depot maintenance and remanufacturing applications and applied to the C-5A application. Several implementation options suitable for insertion into a variety of depot applications that support the C-5A heavy airlifter are described.

Copyright ©1994 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

The Aircraft Directorate at the San Antonio Air Logistics Center (SA-ALC) is responsible for depot level maintenance on the C-5 airframe. The efficiency and productivity of many of the required repair processes will benefit from the insertion of telerobotics technologies. Small batch size, feature uncertainty, and varying workloads make hard automation impractical for a wide range of depot processes. Systems are needed that can bridge the gap between manual control and complete automation. Supervised autonomy and shared control technologies provide intermediate solutions where the human and machine collaborate to perform tasks. In supervised autonomy, robotic tasks are interactively developed by the operator and then executed autonomously [2]. In shared control, control inputs during task execution are provided both by an operator, e.g., using a hand controller, and an autonomous system [3]. A more complete description of telerobotics systems can be found in [4]. The goal is to augment, not replace, the human operator by blending human skills with robotics capabilities.

Aircraft depot maintenance and remanufacturing provides a wide range of challenges for robotics. The physical scale of the applications includes stripping paint from a C-5 heavy lifter to remanufacturing small individual parts. The parts generally arrive individually or in small batches

and a wide variety of parts are remanufactured. Due to the wide variety and scale of the applications, the maintenance and remanufacturing is now done almost exclusively manually. Example depot applications which the architecture must apply to include: painting of the C-5A exterior in a dedicated hanger facility; painting of removed piece parts in a robotic workcell; stripping of paint from the C-5A exterior in a dedicated hanger facility; surface finishing in form of removing material from patches and polishing metal to a high gloss finish in a robotic workcell; Surface cleaning of removed parts in a robotic workcell through application of bicarbonate of soda particulate stream; and sealing and desealing of aircraft fuel tanks.

It is expected that telerobotics can provide many benefits to aircraft depot maintenance and remanufacture. Limited manpower resources limit the number of aircraft that can be remanufactured. Telerobotics can augment the productivity of operators allowing a greater rate of aircraft throughput. In many instances telerobotics can provide better process control, e.g., paint can potentially be sprayed on an aircraft more uniformly than by an operator leading to reduced average thickness and cost savings in paint and aircraft weight. There are various hazardous work situations and environments in aircraft depot maintenance and remanufacturing areas including: chemical contaminants in the air and on shop surfaces; handling large, bulky support equipment; excessive vibration, especially of hand-operated equipment; and excessive atmospheric heat and humidity (up to 100 deg.F, 95% humidity). Telerobotics allows placing a manipulator in the hazardous environment and moving the operator to a safe and comfortable operator control station. Additionally, there are tedious applications which cause fatigue and subsequent errors, e.g., paint stripping and deriveting. Many of these tasks can be accomplished with the operator supervising a telerobotic system to perform the task resulting in greater efficiency and quality.

Since the telerobotic architecture was designed for use across a wide variety of depot aircraft and maintenance and remanufacturing applications, there are a large number of requirements it must satisfy. The architecture must accommodate different types of robotic manipulators with varying degrees of freedom with modular changes only to interface code. It must accommodate different types of transport and positioning devices for robots and piece parts with modular changes only to interface code. Initialization and monitoring must be automated and rapid. Human operations shall be able to safely operate within the range of motion of most manipulating and positioning devices through built-in safety protocols (hardware, software, and/or procedural) Smooth transitions to manual workaround modes must be possible during automation downtimes for maintenance, upgrades, etc. The architecture must accommodate different, unmodeled parts in all piece part applications. Software and hardware upgrades shall cause minimal down time.

2. Example Application: C-5A Aircraft Maintenance and Remanufacturing

The remanufacturing processes that support depot level maintenance of C-5 aircraft are representative of a wide variety of dual-use applications. Applications include stripping the external surface paint and then repainting, painting removed parts in a robotic workcell, skin repair, surface cleaning of removed parts through application of bicarbonate of soda particulate stream, surface finishing for patches, and polishing metal surfaces. A unique aspect of working on large airframes (the C-5A is over 247 ft. long and with a wingspan over 222 ft) is the requirement for large positioning systems. Several alternatives are possible. The first option is the telecrane concept where a special facility provides telecranes upon which the manipulators are mounted, as shown in figure 1. Such a telecrane facility is presently used at Kelly AFB which positions human operators around the aircraft for servicing applications (paint stripping with plastic beads). The telecranes do not have positioning sensors so either positioning sensors would have to be added, or some other method would have to be used to determine the position of a manipula-

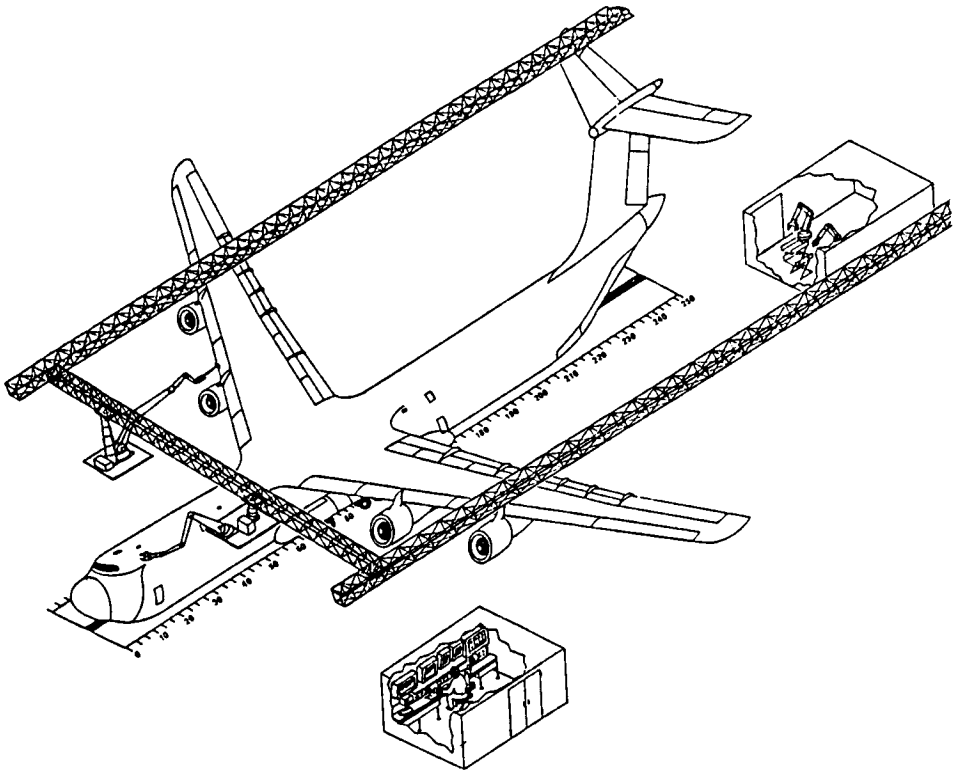


Figure 1: Telecrane concept

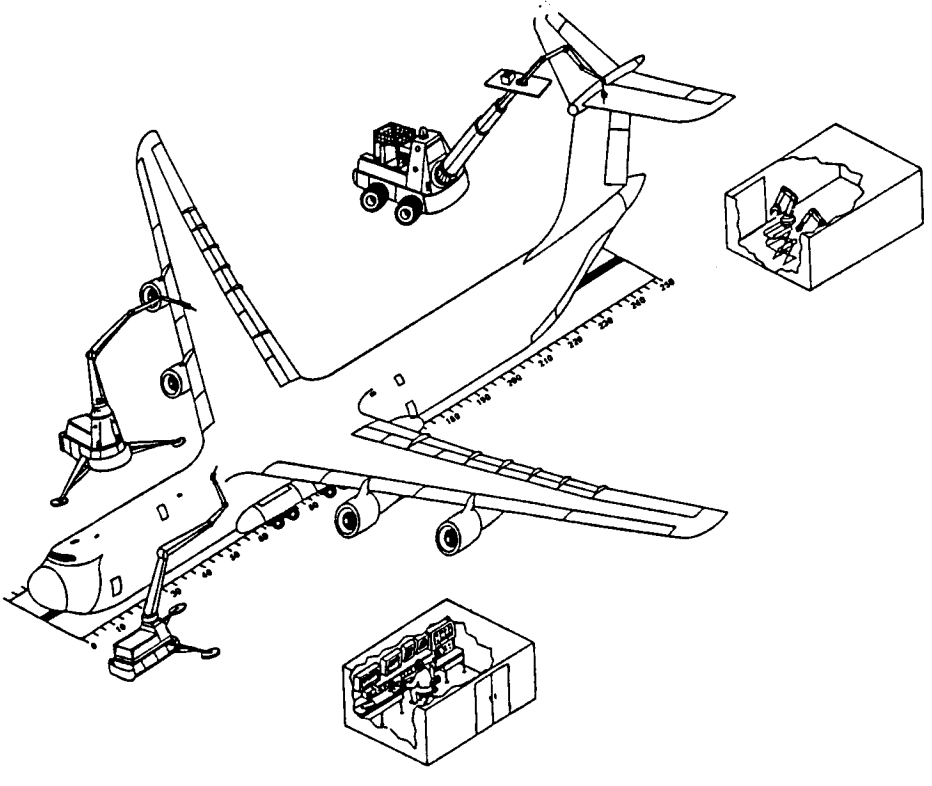


Figure 2: Mobile carrier concept

tor mounted on a telecrane. A second approach is to use mobile carriers where manipulators are mounted on mobile bases and the mobile bases are capable of being positioned around the aircraft, as shown in figure 2. Another option is to use an overhead gantry system where manipulators are mounted on mobile gantries. These transport methods apply to tasks which are done on the physical aircraft structure, such as painting, but there are also many tasks which are done on piece parts in separate workcell rooms such as repair and painting.

3. Telerobot Architecture

A telerobot architecture was developed to provide a near-term solution for implementation of a telerobotics system for C-5A servicing. Various architectures were evaluated such as the DOE GIST architecture[5], the NIST NASREM architecture[6], and the NASA/JPL local-remote architecture[2]. The architecture developed here has ideas common to all of these architectures. The GISC architecture provides the important concept of intelligent subsystems. The NASREM architecture provides valuable contributions in the coordination of task decomposition, modeling, and sensing. The NASA/JPL architecture provides the valuable concept of independent data-driven software modules to collectively provide general task execution capability.

The architecture developed for aircraft maintenance and remanufacturing is shown in figure 3. The architecture is nominally separated into local and remote sites corresponding to the location of the operator and robotic systems, respectively. The actual computing resources can be physically located near the operator, robotic system, or separate from either. The primary constraint is that sufficient communication bandwidth is provided. The basic concept of the near-term system is that there exist subsystems which have sufficient inherent capability to execute a wide range of task types either independently or in coordination with other subsystems. A task program is generated by the local site which describes the task to be executed either by an independent subsystem or through coordination of subsystems. The task program can be executed in various ways depending on the level of capability of the coordinating subsystems. The desired solution is to allow distributed autonomous control of the coordinating subsystems by separating the task program into subsystem task programs. The subsystem task programs can then be executed by a task program sequencer, possibly at the local site operator control station, or sent to the subsystem controller for execution within the subsystem controller, if possible. Subsystem inherent capabilities are programmed offline so that during task setup and execution the subsystems already have the necessary inherent task execution capabilities.

Various maintenance and remanufacturing scenarios provide a poorly structured environment so that sensing the environment is necessary to generate or update a model of the environment. For example, neither the manipulators nor the aircraft will be positioned accurately to a well known location a priori to task execution. Before, or during, task execution, the relative positions of the manipulator and aircraft area of interest must be determined. A main object knowledgebase is provided which stores global state information. Each subsystem also has its own database which includes relevant information from the object knowledgebase and information generated from sensing the environment during task execution. The object knowledgebase and subsystem database are kept consistent for common information. Environment modeling can be done in various ways. Autonomous subsystem tasks can include, or have primarily, modeling elements. Alternatively, the operator can interact with the system to aid in developing models of the environment. For applications which require highly accurate positioning, such as deriveting, it is likely that either sensor based position servoing or shared control will be necessary. An a priori generated model of the rivet pattern is unlikely to have the accuracy relative to the real rivet pattern that would be necessary for rivet removal. Sensor based position servoing would likely utilize real-time vision with an arm-

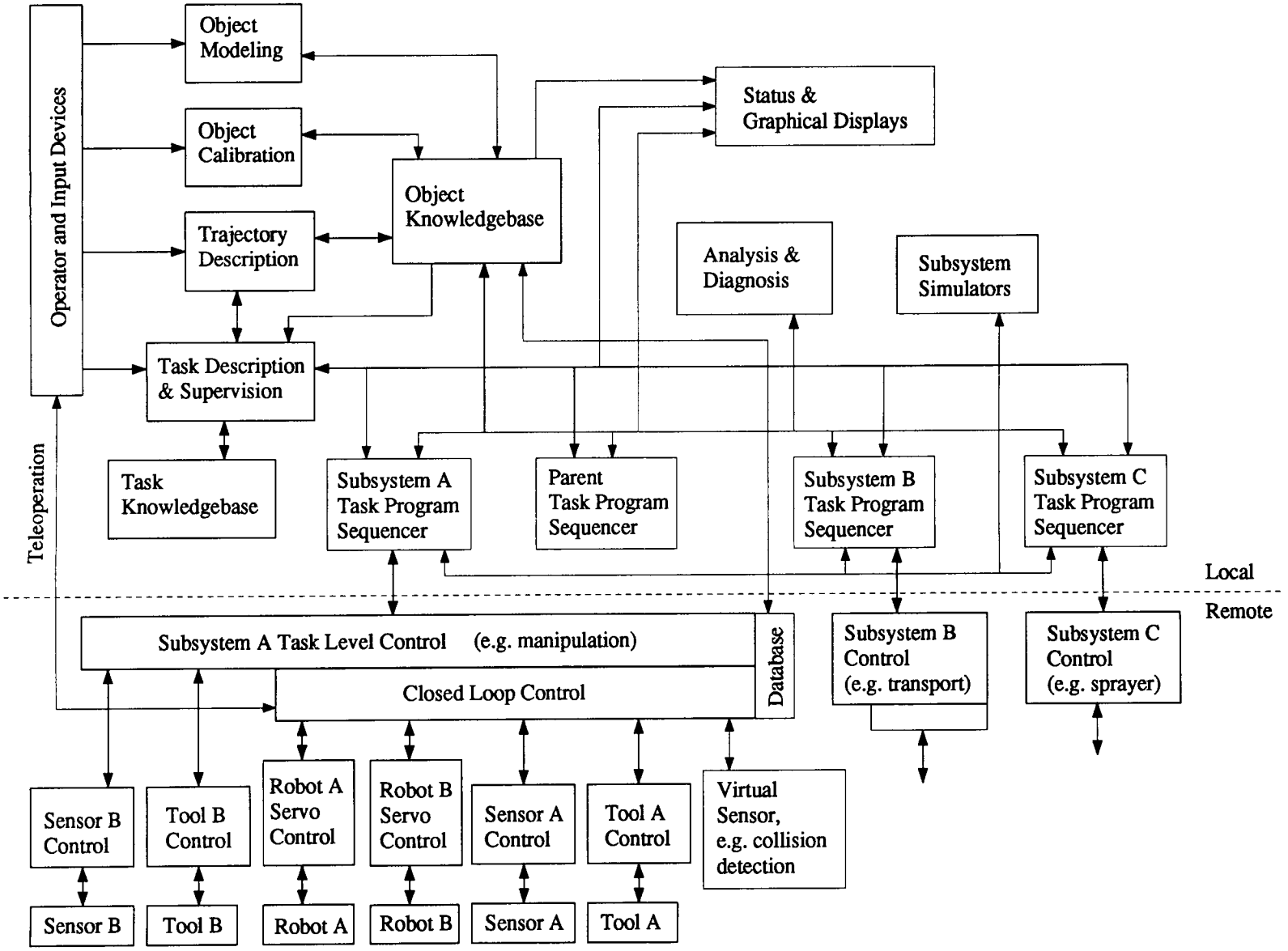


Figure 3: Telerobot architecture for aircraft maintenance and remanufacturing

mounted camera. A proximity sensor and possibly a sensor to measure surface tilt might also be used concurrently to control the position of the manipulator relative to the target. Rivets are difficult to find autonomously since the rivets have approximately the same color as the skin. Also, for previously repaired skin sections, the rivet pattern may not be known a priori. Therefore, for rivet removal, the operator can facilitate the use of the automated vision and sensing system by designating the rough location of the rivets to be removed. A video image of the skin section is provided on a monitor for the operator. If a model of the skin section is available, then it is overlayed on top of the video image (a ghosted image or perhaps wire frame). Otherwise an approximate model of the skin is generated to provide a three dimensional surface upon which to designate rivet locations. The operator then utilizes an input device to move a cursor to the rivet locations seen on the video image and selects the rivets to store their locations in the object knowledgebase. These approximate locations can then be used as starting locations for the automatic sensor based localization later. It is often useful in the task programs to specify objects and locations symbolically rather than with absolute locations. Then the task program can be generated independently of the actual locations. The actual locations of objects can be generated later either independently and stored in the object knowledgebase or as part of the task where operator input is automatically requested. Shared control can also be used to specify destinations. Here the operator uses an input device such as a six DOF hand controller to position the manipulator above the rivet. The proximity and/or tilt servoing could be occurring simultaneously to control the distance to the surface, depending on the method for removing the rivets. In this case the operator replaces the vision system.

It is desired that task description be as simple as possible for the operator. Therefore, as much intelligence as possible is designed and programmed into the system. For a sophisticated implementation, the operator would provide high level goal based information and the system would autonomously generate the associated task programs. A more realistic near-term system would require greater interaction with the operator to develop a new task. It is desired that the operator interact with the system primarily within the video/graphical environment, i.e., in a telepresence sense, both for task description and task execution. For task description, the operator would move the graphical manipulators via an input device such as a six DOF hand controller. The objects to interact with could be selected directly, or implied by proximity or context. The tool which the manipulator is carrying, along with the previous task steps and the selected object, provide a large amount of context information which the system could use to automatically suggest to the operator, or select, the next action to take[2]. The actions could be the subtask segments from the task knowledgebase.

The remote site subsystems will vary in the types of systems which they will control, in capability, and in vendor source. For some subsystems the task program will have to be translated into its command language. For other subsystems, a task program might be used directly. There are several types of control and coordination which may be needed within subsystem control and between subsystems. Closed loop control implies that there is a close coupling between sensory data and control commands to the devices. One subsystem provides cooperative control of its associated devices. Multiple subsystems can be coordinated to achieve a task goal.

Configurations of the architecture shown in figure 3 for telecrane, mobile carrier, and gantry applications are shown in figures 4– 6.

4. Evolution of the Telerobotic Architecture

The architecture shown in figure 3 supports near-term system development and evolutionary growth. Most of its basic features can be provided by existing vendors of automation and robotics technology. One drawback of current technology

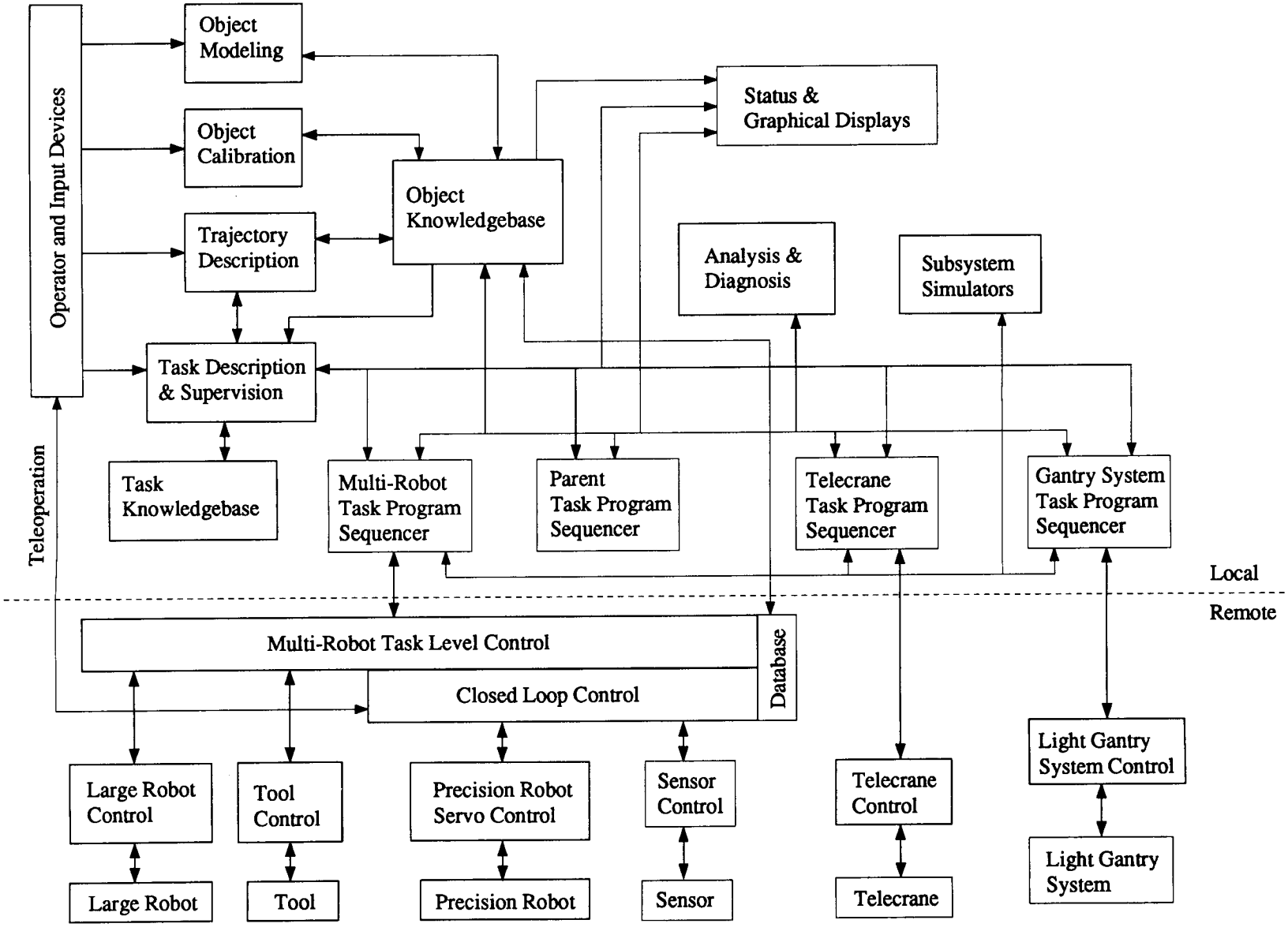


Figure 4: Focused Telecrane Control Architecture

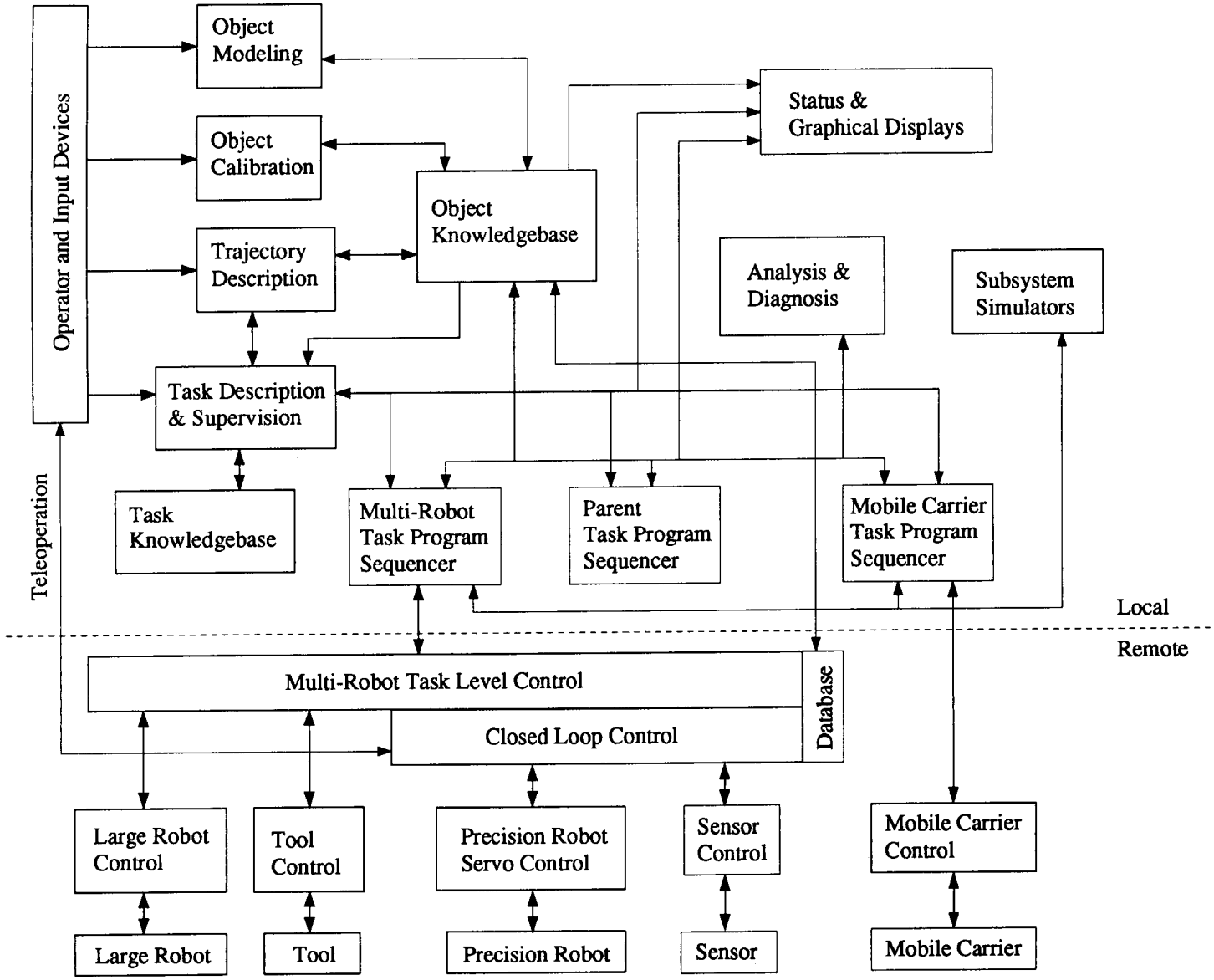


Figure 5: Focused Mobile Carrier Control Architecture

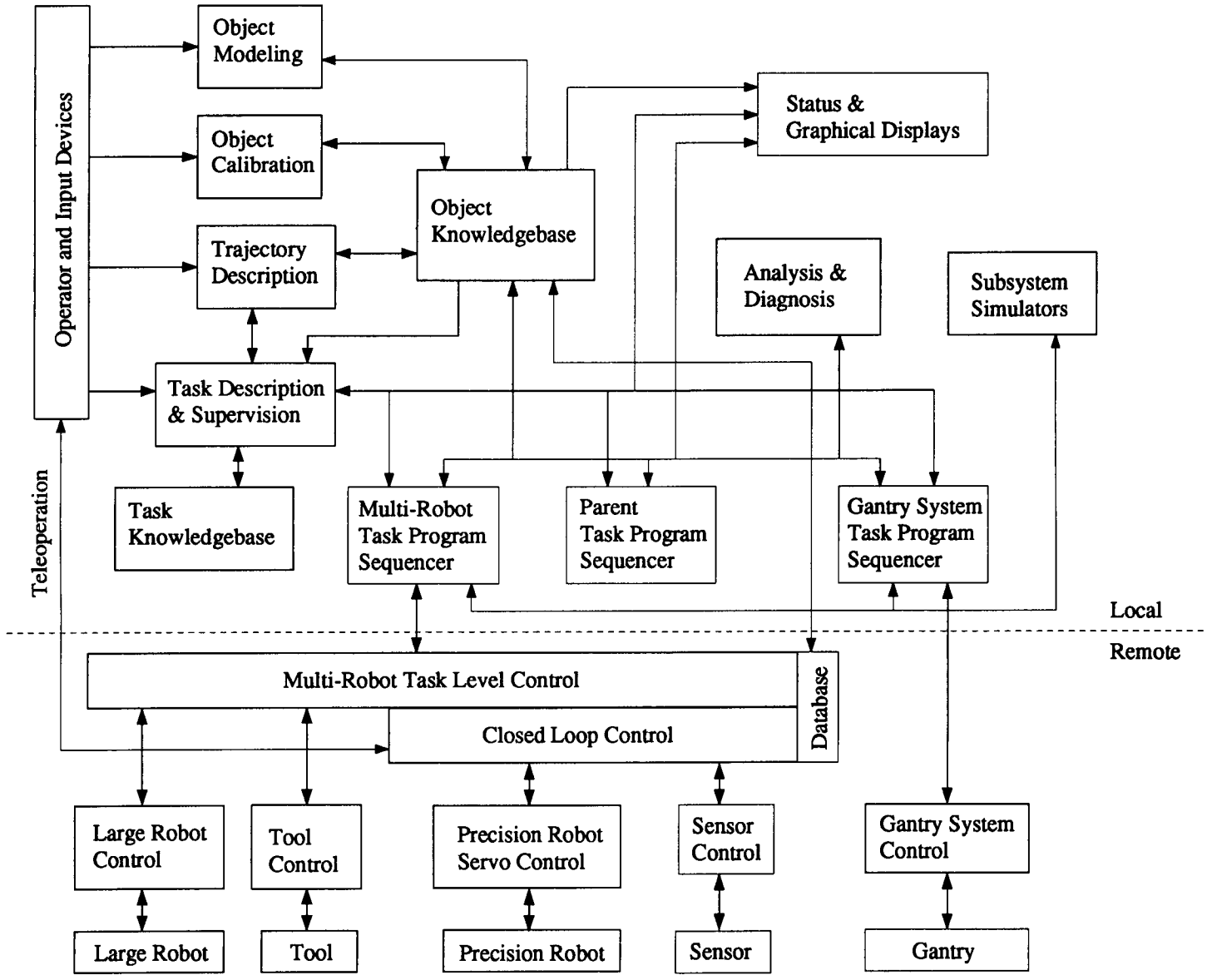


Figure 6: Focused Gantry Control Architecture

is that it is difficult to integrate systems from different vendors when a significant amount of control and modeling information is passed between layers in the architecture, since this information is often stored in different formats. The evolutionary direction of the architecture is to provide subsystems with increasing levels of intelligence which can be provided with goal based information rather than control based information which is prevalent with current technology. The intelligent subsystems would then autonomously request resources from other parts of the system such as the object knowledgebase. The resulting task programs would then be significantly smaller and quicker to generate. Protocols for communicating requests and information between the subsystems need to be developed. This approach is consistent with the goals of the Next Generation Controller program [7] which is developing a similar architecture for machine tool control. In the next year this effort will work more closely with the NGC effort to attempt to develop common interfaces and a common evolutionary architecture. The operator remains an integral part of the evolutionary intelligent architecture. In such an architecture the operator could become one subsystem with multiple capabilities or could be modeled as multiple subsystems. Also, the operator could act as one part of one of the subsystems such as the case described above where the operator performed the visual servoing for rivet localization. The system would then request input from the operator for information it cannot generate automatically, just as it would query one of the other parts of the system.

5. Conclusions

Application of telerobotics to aircraft depot maintenance and remanufacturing was discussed. The requirement to reduce technology insertion and system life cycle costs mandated the design of a generic architecture which can be implemented in the near-term and and still provide an evolutionary growth path. Most of the basic features of the near-term architecture are available from existing vendors. The evolutionary architecture utilizes increasing intelligence in the various modules of the system resulting in a more distributed autonomous control system. A commercialization study is underway.

Acknowledgements

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the Air Force Material Command (AFMC) Robotics and Automation Center of Excellence (RACE).

References

[1] Wayne Zimmerman. A generic telerobotics architecture for c-5 industrial processes. Technical Report (internal document), Jet Propulsion Laboratory, 1993.

[2] Paul G. Backes, John Beahan, and Bruce Bon. Interactive command building and sequencing for supervised autonomy. In *Proceedings IEEE International Conference on Robotics and Automation*, Atlanta, Georgia, May 1993.

[3] S. Hayati and S. T. Venkataraman. Design and implementation of a robot control system with traded and shared control capability. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1310–1315, 1989.

[4] Thomas Sheridan. *Telerobotics, Automation, and Human Supervisory Control.* M.I.T. Press, 1992.

[5] R. W. Harrigan. Intelligent system controller for remote systems. In *Proceedings International Symposium on Robotics and Manufacturing*, pages 741–747, Santa Fe, November 1992.

[6] R. Lumia. Space robotics: Automata in unstructured environments. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1467–1471, 1989.

[7] Bruce M. Anderson and Richard G. Holland. An open standard for industrial controllers. In *Proceedings International Symposium on Robotics and Manufacturing*, pages 953–960, Santa Fe, November 1992.

A Practical Method of Reverse Engineering and Automatic Path Programming for Robotic Surface Finishing

William T. Adams, John M. Fitzgerald, T.J. Lawley*, O. R. Mitchell†
The University of Texas at Arlington, Automation & Robotics Research Institute(ARRI)
Fort Worth, Texas

Abstract

This paper presents a new method of automatic path planning and trajectory generation for robotic surface finishing. Initial development is on a platform integrated from commercially available components including AutoCAD and a low cost CMM. Automatic program execution is demonstrated using a 6 DOF manipulator on a variety of complex shaped surfaces.

Introduction

The focus of research described in this paper was determined by the needs of end-users associated with The Automated Surface Finishing Consortium. The mission of the Consortium is to develop finishing automation as an end-user technology for U.S. manufacturers. Members are military and commercial manufacturers, universities, federal labs, and commercial suppliers of abrasive process technology and robotics technology.

An objective measure of technology development/deployment impact is the availability of successful commercial off-the-shelf technology. The synergistic interactions of the Consortium members have yielded impressive results to date. Force controlled robot end effectors commercialized by collaborating members have significantly broadened the boundaries of available compliant finishing process capability. A generic off-the-shelf integrated robotic finishing system, the product of another collaboration, is now distributed and supported by the largest robot company in the world. Likewise a low cost high performance simulator based surface path programming aid is now a commercial product. ARRI is host to the Consortium, and a member.

Background

Most material forming processes do not produce finished parts. Most machining operations leave burrs and sharp edges. Large aircraft wing skins are milled by three axis terrace cutting leaving small steps which must be blended to prevent fatiguing stress concentrations. Complex curved surfaces like ship propellers and aircraft landing gear are machined with rounded milling tools which leave a pattern of tool marks which must be ground off by hand. Cast parts require gate and sprue removal and deflashing. When castings are machined they require deburring. Many parts must have their surfaces conditioned for appearance or subsequent plating and coating operations. The stamping and forging of automobile door panels and engine components leave "imperfections" which are finished out by hand. Die cast surfaces of hardware for door handles and hinges are ground and polished. Wrenches, golf clubs, vacuum cleaners, furniture, boat hulls, all have to be ground, sanded and polished to have a nice appearance.

The costs of manual finishing are high. A recent report by the National Center for Manufacturing Sciences suggests that direct finishing costs exceed 25 billion dollars annually based on mechanical manufacturing industry gross sales of 1,000 billion dollars[1] . Often in precision parts manufacturing and particularly with complex shaped parts there is high finishing labor content. Jet engine component manufacturers in the U.S. and Canada report that between 15% and 30% of the direct cost content of their products is finishing labor. A major consequence of manual part finishing is scrap and rework. Although difficult to quantify, interviews of finishing workers in high value added manufacturing invariably reveal costly scrap and/or complicated expensive rework

*Professor, Department of Mechanical Engineering, UTA
† Chairman, Department of Electrical Engineering, UTA

Copyright c 1993 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved

procedures. This is a direct result of manual finishing errors.

Most of the finishing laborers are grinding, filing, sanding and scraping parts by hand. Some finishing jobs are high paying, requiring artisanship based on years of practice. Most finishing jobs are low skill low quality jobs, typically they are the least desirable and lowest paying. The reason is that manual finishing work is drudgery of the worst kind, exposing workers hands and arms to mechanical impact and vibration while requiring repetitive motion, and often producing dangerous and toxic dusts. Fortunately, turnover rates are relatively high in the lower paying jobs which helps to limit prolonged exposure. There is a great need for practical hands-off finishing process capability.

The application of abrasive processes to meet finishing needs is diverse and widespread in manufacturing. The abrasive industry, from mining through tool design, manufacture, and distribution, is a multi-billion dollar per year industry. The number of permutations of tool size, shape, abrasive type, and other characteristics of manual abrasive tools is astronomical. Finishing processes are typically performed manually, the process is planned based on knowledge and experience, and controlled by the operator with his senses of sight, touch, hearing and even smell. Desired output is often not measured but judged. Automatically controlling finishing processes in the same way as human operators control them would be difficult to achieve in practice. Other methods of automation do work.

Scores of robots are now deployed in U.S. factories finishing a wide variety of parts[2]. There are many factors which have contributed to the growing commercial success of robotic finishing. The dominant success factor is the pioneering effort by 3M and others to develop the compliant abrasive finishing processes. The key is compliance between part and tool, critical because it reduces variation of tool force, a dominant process control parameter. Even slight variations on the order of a few thousandths of an inch in the relative location of the robot's commanded tool point path and the actual contact point will result in large changes in tool force reactions if the system is too rigid. Normally occurring tool wear, part-to-part geometric variation, manipulator kinematic error, and set-up errors drive rigid abrasive processes out of control. There is a wide variety of abrasive tools available which are by the nature of their material composition, naturally compliant. Also, several constant force devices are commercially available which provide added compliance in one direction, usually normal to the surface of the part. With constant force end effectors, more aggressive abrasive tools can be used with robots.

The Need for Automatic Programming

A major barrier to further finishing robot deployment is system programming. Generating the robot motion control sequence is a major problem in manufacturing operations which produce parts in a variety of complex shapes. One manifestation of the programming problem occurs when a variety of complex shape parts must be finished. Each part type requires only one program which is executed repeatedly for its specific part type, but the programs have a large number of taught points. An example is the polishing of large asymmetric shaped aircraft skin panels. One manufacturer determined that manual teaching of robotic polishing programs was to too costly when their part mix approached eight to ten part types.

In cases requiring a unique motion sequence to finish each individual part, manual methods must be used because of the lack of practical programming methods. For example in aircraft remanufacturing, aerodynamic surfaces dented in normal service are routinely repaired by applying filler materials which are ground to shape by hand. Since the location and extent of this type of damage is random, each part requires a unique grinding program. Unique plans are commonly required for finishing turbine blades and propellers. The location and amount of excess material left by machining and forging is not entirely predictable. An automatic programming system which can be integrated with commercial robotic surface finishing equipment is needed.

Automatic finishing with compliant abrasives does not require automatic process control and planning. Planning is uncomplicated for many applications because process settings such as abrasive type, tool cant angle, tool force, tool feed, and tool speed remain constant for most or all of the individual job. Path planning and the subsequent trajectory generation still require too much manual effort for many robotic applications.

The approach taken here to developing automatic programming for robotic finishing is to first automate the path and trajectory generation

while depending on the human operator for higher level process planning, supervision and control Interviews with end user companies revealed that using CAD design data as a basis for modeling surfaces for path generation purposes would be impractical. So reverse engineering the surface to create a geometric model was included as a system requirement. For reasons of maximizing practical application success, commercial products are used to the greatest extent possible. Furthermore, overall system cost is minimized so that smaller companies will more readily be able to acquire the technology. This approach serves the objective of providing valuable off-the-shelf programming automation while providing a basis for the development of intelligent and more fully automatic processing.

System Equipment

A personal computer(PC 486-50mhz) and Faro Metrecom Coordinate Measuring Machine(CMM) are used with MS-DOS and AutoCAD as the platform for the automatic path and trajectory programming system. The Faro Metrecom CMM is a 6 DOF spatial digitizer interfaced to AutoCAD. AutoCAD's 3D modelling capability allows creation of, access to, and modification of geometric objects through AutoLISP and C. Customization of AutoCAD's environment through menu building using AutoLISP and Macros makes the objects accessible and easy to manipulate. The Fanuc model S-700 is a 6 DOF serial link manipulator with multi-tasking capabilities using the R-J controller. The R-J controller is off-line programmable. The Karel language programs can be written and compiled on a PC workstation. The R-J controller interfaces to a PC from which compiled robot control code can be accessed. The source code is written in ASCII format in Karel, then it is formatted and translated to executable code using the Karel translator program supplied by Fanuc. A server supplied by Fanuc is used to download the translated code to the controller. The PushCorp constant force end effector which was developed as a compliant abrasive end effector for surface finishing has a dedicated controller with interface for the R-J controller. The end effector mounts directly onto the faceplate of the robot with the axis of rotation of the grinding disk parallel to that of the faceplate.

Setup Procedure

Typically surface models for off-line programming applications are generated using constructive solid geometry (CSG). In this case the geometry will be recreated in a CAD environment from digitized surface location data in a process called "reverse engineering". Before digitizing the surface a common coordinate system is established between the CMM and the robot.

The robot can relate a tool frame to a base frame and provides several programmable frames of both types as well as one predefined frame of each type. Since it is the location and orientation of the grinding disk that is of importance a tool frame is taught at the center of the disk with the CFD positioned at the midpoint of its compliance stroke. There are different methods of teaching a tool frame to this robot through the teach pendent menu, and the method used here is the 3 point method. The desired tool point is placed at some fixed point in space which is taught three times with three completely different end effector orientations. This point on the tool can then be moved in space to any point within the robot's workspace relative to the base coordinate system or a user defined coordinate system.

Next, a user defined coordinate system is defined using a 3 point method for the robot and the CMM simultaneously. First a new origin is taught to the robot, and then the same origin is taught to the CMM by placing its tool point at the tool point of the robot and recording it. The robot is then moved parallel to the X axis of the base coordinate system several inches away and taught a point on the new X axis which is followed by teaching the CMM the same point to define it's X axis. Finally a point is taught to both devices on the positive part of the XY plane. The CMM automatically translates to this new system, and the robot can be instructed to do so through the teach pendent or through off-line programming tools.

This entire process can be done in 10 minutes and is only necessary for initialization or when the set-up is disturbed. Once the two devices share a common coordinate system the Faro CMM can be used to generate a CAD model of the surface to be finished, and the coordinates on the surface will be the same with respect to the CMM or the Fanuc Robot.

Surface Model

The Faro/AutoCAD interface allows the user to move the endpoint of the CMM digitizer through some trajectory in 3D space and generate a 3dpolyline entity with complex entity nodes at a predetermined interval. By keeping the endpoint of the digitizer in contact with a contoured surface, a 3dpolyline that lies on that surface is created.

AutoCAD has the capability to generate a Coons Patch based on four bounding lines in space that are connected at their endpoints. The number of facets, or mesh segments are programmed using the variables surftab1 and surftab2. The importance of this is that facets will be used to determine the path node frames of the end effector trajectory. Mesh density in the direction of travel will affect the smoothness and accuracy of the motion while mesh density in the direction perpendicular to the direction of travel controls the spacing between passes.

An AutoLISP program is activated through the customized pull down menu at which time the Faro 3dpolyline programs are executed and request the user to drag the digitizer over the desired boundary. This operator defined boundary encloses the region on the surface to be finished. The program then automatically performs the required modifications of the polylines to ensure that they are connected at their endpoints. The result is four 3D lines in space connected at their endpoints forming the bounding edges of the Coons Patch surface of m by n surface facets. Upon completion of generating the four boundary lines and assigning the entity data lists to variable names, an interpolated surface is automatically generated using the Edgesurf command. This command requires selection of the four existing bounding edges and is answered by the program with information from the stored entities. The direction of travel is requested upon completion of the mesh generation process and can be in the m or n directions.

Extracting Tool Pose Information from the Surface Model

The problem of determining robot tool poses from surface data involves several steps. A brief outline is helpful in describing the work that has been automated.

1. Create or access an existing surface

A function is implemented to number the nodes of the mesh and add this integer data to the actual entity data which was created by AutoCAD. This is done to simplify the development of an algorithm to access individual nodes and their immediate neighbors on the surface.

2. Determine a unit surface normal vector

This is done by averaging several normal vectors in the region. Each node P being considered as a path node has four immediate neighbors including two in the direction of travel and one each on either side perpendicular to that direction. These will be referred to as P_1, P_2, P_3, and P_4. The cross product between vectors originating at P and terminating at two of the neighboring nodes yields an approximate normal vector to the surface at that point. An algorithm is implemented that performs this same operation for four quadrants of the irregular quadrilateral formed by these points with P as the origin for each vector which results in four approximate surface normal vectors at the node P. These four approximate surface normals are then averaged and normalized to get an approximate unit surface normal at the node P. This procedure is performed for each node on the surface excluding those on the bounding edges.

3. Define an angle of attack

When the program that calculates and draws the surface normals is complete, another function is automatically called that requests a cant angle[3]for the tool. This angle is the pitch angle of the tool that will be used for the finishing process. The user can type a number in degrees at the command line after which the information is used to calculate tool frames at each node of the path.

The calculation of tool frames is based on the transformations found in Craig's Text[4]. The result is shown graphically on the screen in the form of a coordinate system at each node in the orientation that the end effector will be in at that node in order to maintain the desired cant angle with respect to the surface and the direction of travel.

4. Generate the Karel Program

At this point the tool frame information must be expressed in the language understood by the

robot. The user is prompted for a filename which will be used to store a Karel program in the form of *fname*.kl and a data file in the form of *fname1*.kl , *fname2*.kl and so on for each branch. Another program then implements an algorithm developed to determine which nodes belong to a branch of the path, and builds the path as a list containing branches which them selves contain lists. A branch list is of the form (1 5 6 7 8) where the first number is the branch number and determines it's order of execution among branches, which is followed by the node numbers which define which nodes will be traveled to in this branch and their order. Each branch begins at a boundary and follows the mesh in the previously chosen direction until it reaches the opposite boundary and will include all nodes along the way that are not actually on the boundary. The initial and final nodes are extrapolated linearly based on the two first or last nodes respectively with a user defined offset in the direction away from the part in order to allow a smooth approach or departure to or from the surface. These extrapolated nodes will not show up in the path data list and are calculated just before being written to the data files

The final path variable will look something like this :

(pa1 (1 7 8 9 10 11) (2 18 17 16 15 14) (3 20 21 22 23 24))

and will be used to write the path data files. The path data file contains only information that is relevant to the Robot controller and it's path data structure.

Each branch of the data is written to a data file in a specific format that is shown in the following which was taken from an actual data file.

```
536.6210 103.9143 -208.2082 175.8836 3.6680 -9.1591
479.4422 114.2409 -259.2189 175.8836 3.6680 -9.1591
422.2633 124.5675 -259.4296 178.8781 -5.6877 -11.3613
366.8345 136.8362 -270.6763 -177.4433 -14.8075 -11.1896
312.7229 146.2052 -288.9642 -174.9756 -19.1880 -11.4856
259.5330 158.6093 -308.7990 -173.1321 -20.1050 -13.6317
206.3431 171.0134 -277.8339 -174.9756 -19.1880 -11.4856
```

This data file represents branch 1 from the path list and each line in the file represents the data for X, Y, Z, and the Euler angles required to define position and orientation of the end effector for a particular point on the surface. There are several types of data formats in Karel including the type *path*. A path type variable is a structure which can contain several other specific types of data including position data in the form shown above. Several positions and orientations can be stored in a single path variable, and then used in a move_along statement which instructs the robot to move it's end effector through those points sequentially. There are also motion control statements that can be used in conjunction with the move_along statement that affect speed and acceleration between nodes.

A LISP program is then called that writes a Karel program to be translated and sent to the controller. The Karel program is then loaded into the controller and executed at which time it reads the data files and builds path variables in a routine called build_path. Each branch of the path is built before being used in motion statements that cause the robot to follow them. The move_along statements are executed sequentially and the tool moves across the surface as planned.

The actual grinding process can be initiated either through the Karel program by way of an option included in the custom menu, or by using the teach pendant.

Conclusions

A practical method of automatically generating tool point paths and robot trajectories for surface finishing has been developed and implemented using off-the-shelf technology components. This method is expected to only increase average robotic finishing system costs on the order of 5%-10% of total cost. High level process planning and supervision remain under human operator control.

More research is needed to develop practical process planning and control methods to supplement the operators process knowledge so that new applications requiring new processing recipes can be quickly implemented. Also sensor applications must be developed which augment the operators ability to assess surface condition and geometry.

A longer range objective is to develop automatic finishing as a smart processes which can be driven using feature based CAD data from intelligent high level supervision and control systems. Automatic processing capability must be broadened beyond the compliant abrasive process methods used today.

Efforts to build the manufacturing information technology infrastructure to support intelligent agile production continue, but the automatic planning and control of most shop floor processes including finishing is not adequately understood. Furthermore, the application of sensor technology needed for automatic observation to support finishing process control is not understood. Much empirical work will be done to discover practical process planning, control, and sensing methods for automatic finishing. The automatic path generation capability developed through this research will immediately improve application development productivity.

Acknowledgments

The research described in this paper was conducted in part with funding from Vought Aircraft Co. Through the ARRI membership program in ARRI's finishing laboratory. Vought also supplied complex shaped fuselage skin and fixturing. Tim Graf of 3M provided valuable information on end-user path programming needs. 3M's Abrasive Products Division donated all abrasive tools. The FANUC S-700 robot was consigned by FANUC Robotics North America, Inc. PUSHCORP. Inc. of Dallas, TX supplied the force controlled end effector and donated valuable technical support. FARO consigned the CMM.

References

[1]National Center for Manufacturing Sciences, Automatic Inspection Systems for Edge and Surface Finishing, Technical Report NCMS-88-MPM-9, April 1989.

[2]Tim Graf. Deburring, Finishing, and Grinding Using Robots and Fixed Automation: Methods and Applications. In *Proceedings International Robots & Vision Automation Conferenc*, pages 20.1 - 20.22, 1993.

[3]James C. Davis, Path Planning Methods for Robotic Grinding of Complex Contoured Parts.MS thesis , University of Texas at Arlington, 1993

[4]John Craig, *Introduction to Robotics* Addison-Wesley Publishing Company, 1986.

AIAA-94-1267-CP

Virtual Environments for Telerobotic Shared Control

Brian K. Christensen

Deneb Robotics
Auburn Hills, Michigan

ABSTRACT

Traditional teleoperation depends solely on operator skill for effective task completion. This dependency limits the class of operations which were suitable for teleoperation. In many work situations today, external and operational requirements necessitate a more efficient operational scenario. The potential environments are more complex and dangerous and require integrated safety systems. One way to enable more effective control is to share control between the human operator and an intelligent computing system. This paper discusses the requirements and an implementation of a virtual environment for telerobotic shared control.

Efficient, effective and safe system operation depends on the operator's ability to make wise decisions. In order to make wise decisions the operator needs a clear understanding of the operational requirements, and an understanding of the external factors which affect the operation requirements. A virtual environment built upon a shared control interface can provide much of the information required for safe operation. The virtual world allows the operator to clearly visualize the task and provides feedback that is not otherwise available. The shared control interface verifies operational demands against system constraints. The virtual environment provides for a practice and training environment where mistakes can be made. Only after an operation has been cleared for execution does physical motion take place. Proper linking of human input with computer controlled heuristics greatly increases the safety level of the system.

The knowledge base that the computing system uses to perform decision making is called the *World Model*. Visual information from the *World Model* is displayed graphically as rendered, shaded, texture mapped animated polygons. The core of the shared control system is the computational engine that maintains, displays, visualizes and controls interaction with the *World Model*. The computational engine can be interrogated at several different levels. During operation: multilevel interaction defined by the program (mouse, keyboard, multi-modal input devices). The visualization system offers a high interactive operating environment, enabling users to change objects, display attributes, and elements of the *World Model* in real time.

High speed and optimal interaction with external input and teaching devices are required for effective interaction. Traditional teleoperation allows the user to effect immediate changes in manipulator configurations via an input device(s). Anything from a simple joystick to a force reflecting master have been utilized. The virtual environment uses these input tools and more advanced tools such as datagloves to direct user input into the graphics environment. This enables the use of the graphics environment for training, simulation, design interactions and will allow the *World Model* to interact and interpret operator commands.

2. INTRODUCTION

The integration of a graphical-based programming and control environment has the potential to enhance use and utility of robotic systems. Traditional off-line robotics programming methods address some of the needs for increased productivity of robotic systems. A more complete solution will address simplifying the programming process, direct linking of the graphics and control environment, the needs of cell calibration and model registration, and the needs of sensor- directed robot control. The graphic model part of the solution allows for extensive planning and conceptualization to occur without the use or need for actual equipment. The direct control side of the system allows for the seamless communication of commands and information from the graphical model to the device controller, and from the device controller and system sensors back to the graphics model. The sensor-based control requirements and the inclusion of calibration methodologies allow for confidence that the modeled world corresponds to the actual environment. The research described here represents our efforts to design and construct complete tools for enhancing robot or intelligent device planning, programming, and control processing.

Copyright © 1993 by Deneb Robotics. Published by the American Institute of Aeronautics and Astronautics, Inc. with Permission.

3. HAZARDOUS ENVIRONMENTS: A TECHNOLOGY DEVELOPMENT FOCUS

An application focus is an important element of technology development. An application focus not only provides an understanding of which technology areas should be addressed first, but frequently suggests which technology approaches will be the most fruitful. For example, the highly structured nature of most manufacturing environments fostered the development of pick-and-place robots and strongly suggests that fixtures, not real-time sensing, is the most appropriate way to locate objects in the robot's work space. Teaching, in which the operator manually moves the robot to a location in the environment and stores that location (usually based upon encoder readings) in the robot's computer memory is the most common form of robot programming in manufacturing environments. During operation, the robot is instructed to repetitively return to some predetermined sequence of these previously taught locations. Fixtures assure that the workpiece is where it is expected and the robot carries out its action blindly. Production rates, not safety, are of primary concern. Personnel safety is typically provided by excluding people from the robot's workspace. If a part's location is not correct and an accident were to occur, the part or the robot might be damaged, but extensive damage is not expected.

Hazardous environments, however, place a premium on accident-free operation due to potential serious consequences of damage to the workpiece or environment. Rather than assuming that all objects in the robot's workspace are where they are expected, as in manufacturing environments, objects are always anticipated to be in unexpected locations in hazardous environments and all robot motions must be continuously monitored and validated.

Much of the original work leading to the development of graphical programming technologies at Sandia National Laboratories was performed to enable the application of robotic systems to hazardous nuclear and toxic environments. Due to the desire to limit personnel access, it is desirable to program robot systems within the context of a model of the environment, rather than use the teaching approach which would require operator entry into the hazardous environment. Without operator entry for close-up observation, teaching-based manual programming is very difficult. Thus, it is desirable to reduce the need for detailed operator programming and to transfer much of the reasoning task to the computer system (including sensors) controlling the robot. The knowledge base which the computing system uses to perform decision making is termed the *World Model*.

To allow the robot's control computer to intelligently reason about the environment, environmental sensing must be provided. Environmental sensing allows dynamic updating and validation of the *World Model* so the computer system's reasoning process is based upon a valid model. Real-time sensing increases operational safety by allowing the computing system to adapt the robot's movements to compensate for uncertainties in the *World Model*. Under no circumstances may the robot be allowed to increase the hazard associated with the already hazardous environment. If this occurred, robots would not be used because safety is an overriding issue when dealing with hazardous environments.

4. DEFINITION OF INTELLIGENT ROBOTS

Since the most common commercial robot applications are for repetitive operations requiring no sensing or decision making, robots are frequently envisioned as devices capable of only this type of behavior. Based upon the discussion above, many hazardous environments clearly require intelligent robot systems with more advanced operational characteristics. Thus, it is worthwhile to define what the term intelligent robot system means in the context of this paper.

Intelligent systems are systems which can make appropriate decisions when presented with situations. Such decisions include selecting motions which accomplish a goal without collisions with objects in the environment. John Hopcroft viewed robotics as the study of representing and reasoning about physical objects in a computer. Incorporating the more traditional mechanistic view of robotics provides the definition used here:

"Robotics is the integration of the sciences of sensing, representing and reasoning about physical objects in a computer coupled with electromechanical systems to carry out purposeful actions."[1]

In addition, intelligent robots possess skills with well characterized capabilities which can be used to accomplish tasks. A skill, for example, might include applying well-formulated knowledge about forces of interaction to perform in-contact operations, such as assembly, without operator intervention.

A reasonable goal for an intelligent robotic system is to serve as a supervised electromechanical system possessing sufficient intellect to serve in the place of a similarly supervised human. In the event a robot manipulator cannot accomplish its task due to errors, it either re-plans or

requests help from the supervisor, much as the replaced human would do. While intelligent robotic systems possessing the full range of capabilities implied by this goal may be beyond the current state of the technology, significant strides are being made.

5. A CONTROL ARCHITECTURE FOR INTELLIGENT SYSTEMS

In order to satisfy the performance characteristics described above, a structured computing system has been developed which allows incorporation of the wide range of capabilities required for the successful operation of robot systems in hazardous environments. The required capabilities range from fast, servo-level response based on sensor inputs (*e.g.*, follow a surface contour based upon interaction forces) to slower responses in which evaluation of various alternatives is involved (*e.g.*, planning a trajectory for the manipulator end point). The computing environment must integrate all levels of control into an efficiently executed control strategy which smoothly transitions from one control mode to the next. The basic computer architecture used for intelligent robot control is shown in Fig. 1. The hierarchical control environment has two main systems; the reasoning system in which high level control processes are performed, and the real-time control system in which fast response control processes are carried out.

Within the reasoning system, the computer constructs an approximate *World Model* based upon knowledge about the environment (*e.g.*, a map), robot characteristics (*e.g.*, kinematics) and heuristics about objects in the environment and their behavior (*e.g.*, physical limitations of the robot). The reasoning system also displays various aspects of the *World Model* (*e.g.*, geometric models, x-y-z plots, parameter traces) for operator understanding and assistance in decision making. As indicated above, this *World Model* is modified by sensory information and provides the foundation for automatically generating plans (*e.g.*, a collision-free robot trajectory) which are translated into robot manipulator motion primitives. The control processes taking place within the reasoning system can be quite complex and may require considerable computing time.

Within the real-time control system, servo control of the robot is accomplished. Responsiveness of the control system is extremely important as it is responsible for executing the robot motions developed by the reasoning system or supplied by direct intervention by the operator. A slow real-time control system would introduce delays between the commanding of movement and the execution of that movement by the robot, leading to instabilities in the system. Sensors and the *World Model* are employed to monitor the execution of these motions and to automatically perturb the robot motions if necessary to provide safe operation while accomplishing the desired task. Situations requiring direct operator control vary from the teaching of robot locations to the recovery from errors with which the robot control system is unable to cope. Experience with master/slave manipulators suggests that even highly trained operators experience considerable difficulty and tedium when executing remote manipulations.[2] Tedium can result in unsafe operation. Thus, direct operator commands are also monitored by the intelligent control system for compliance with safe operating practices and accepted procedures.

Within the context of Fig. 2, the operator takes on the role of planner and develops task plans for the robot manipulator to execute. However, the computing and sensory systems maintain their role as developers of approximate *World Models* and real-time controller. Much in the manner that the real-time control system perturbs the robot commands of the supervisor, the real-time control system now perturbs the commands of the operator within the context of a *World Model* and sensory system. The real-time control system assists the operator in automatically avoiding obstacles and executing controlled interactions with the environment while the operator performs the high-level task and path planning. Such computer assisted approaches to manual operation have proven effective in providing responsive robot manipulator systems capable of safe flexible operation when manually operated. [3]

5.1 Geometric Modeling

The intelligence of an intelligent robotic system resides in the *World Model* and algorithms for accessing and using the knowledge contained therein. The representation of the *World Model* is critical and the subject of much study because the efficiency of information retrieval determines the usefulness of the knowledge. [4] Robots deal with physical objects and, as stated by Hopcroft, reasoning about physical objects is the key element to constructing intelligent robots. A robot's ability to interact with a physical object is, to a large extent, defined by geometries. Effective geometric reasoning is the key element in constructing practical *World Models*.

5.2 The Role of Visualization Models

The *World Model* as described here encompasses much more than a model of geometric objects. The *World Model* is the visualization conduit providing the operator with insights into the performance of the control system and

displaying the results of control decisions. In addition to the geometry of an object, the model includes the kinematic, dynamic and motion characteristics of all movable objects in the environment. The *World Model* contains the knowledge to understand and predict how a controllable device will move and provides insights into how to plan a trajectory or path and in some cases may include knowledge on how to carry out tasks or complex operations. The *World Model* functions as the operator's window to the control of the intelligent system. It acts to validate command operations, provide real-time feedback to the motion or changes in the system, and provides an ideal place for quality assessment and data logging of all operator commands and the results of those commands. The visualization of the *World Model* is the key to operator understanding of system operation.

5.3 Simulation vs. Control

Over the past few years, much work has centered on the simulation of robots. In many cases this was the first step in off-line programming of these devices.[5,6] For static environments this may be all that is necessary. However, many of the control tasks of today are not static situations. These environments are typified by the challenges of remediation of hazardous waste or the manufacturing environment of flexible manufacturing systems. In these operations the work environment is not always well characterized before work begins, and may in fact change as the system operations are executed. The dynamic nature of these environments require a close coupling of the simulation and control systems. As updated information about the work environment is made available, the simulation or planning system must be capable of responding to the updated information and perturbing the commanded operations in response to these changes. The updated system information may be made available from sensors or from changing production requirements for flexible manufacturing systems.

In addition, the close coupling of the simulation and control system allows for safe integration of operator commands into the control of the robotic systems. The visualization system of the *World Model* acts as a filter to the operator commands. The *World Model* tests and in some cases perturbs the command before it is implemented. This is critical for the safe operation of systems requiring operator intervention.

5.4 Graphical Programming and Control

The goals of the development of the graphical programming and control system are faster and safer system operation and enhanced operator programming. The very nature of a 3-D representation of a modeled system enhances user operation, user understanding and aids user visualization. Graphic representations make it easy for the operator to program the system with icon selection and interactive manipulation of the graphic images. 3-D representation also enables whole-body collision detection not available anywhere else. Through simulation of camera views a camera is virtually presented to the operator. These views serve as an enhancement to live camera views, which may be limited and incomplete, and can enhance operator understanding of the work environment.

Control decisions which are deemed safe within the context of the *World Model* are communicated directly to the intelligent devices. Any translation is done on line and as part of the communication process between the reasoning system and the real-time control system. The motions caused as a result of these commands are fed directly back into the visualization system for recording and operator understanding and as a check on the validity of the commanded motions. In addition, sensors present in the intelligent devices locally verify modeled geometry and communicate discrepancies to the *World Model*. The *World Model* is dynamically updated and geometries constructed or modified as necessary. The updated information is then available for all future control decisions.

6. Implementation and System Description

A commercially available graphical robot simulation environment IGRIP from Deneb Robotics, Inc., was combined with the Sandia developed Generic Intelligent System Controller (GISC).[7,8] IGRIP was operated on Silicon Graphics workstations. Thus, we have built upon rapidly improving graphical computers and the large base of computer graphics capabilities. As the state of the art in high speed graphics computers and computer graphics improves, these advanced capabilities can be directly integrated. A critical area of development in the graphical programming of intelligent robots is providing high speed and efficient communication between the graphics *World Model* and the intelligent sub-system control elements. This is essential to allow graphics models to send commands to other processes and to allow other processes to send commands, interrupts and positional information to the graphics *World Model* for model updating and display.

The Low Level Telerobotic Interface (LLTI) developed jointly with Deneb Robotics to allow communication at the binary level between processes was the first attempt to tightly couple external data to the animation display. This

high-speed interface is essential to operator understanding of changes in the actual robot environment. It is also essential for proper visual feedback of operator-directed changes in the graphic environment using external input devices.

GISC allows the operator to control both the simulation system and multiple disparate programmable devices from the same control environment and in the same programming language. This is enabled by the use of the Sandia developed Robot Independent Programming Environment (RIPE) and in the Robot Independent Programming Language (RIPL).[9] The use of RIPL allows the system designer to communicate to many different programmable devices in the same manner. At one level the use of RIPL allows the programmer to communicate and direct the robot from different manufacturers in the same language syntax and structure. At another level through combination of the *World Model* and RIPL language, the operator can program any robot by simply graphically causing the robot model end point to move in response to external inputs. The benefits of the graphics environment is that the operator immediately sees the results of his directions and the operator is warned of potential collisions between the robots and objects in the environment. In fact, motion plans which would potentially cause collision are not permitted to be communicated to the controller and executed.

GISC employs a distributed computing environment for supervisory and real-time control of the robotic system, as shown in Fig. 1. The computing environment consists of a Silicon Graphics Iris (SGI) workstation which runs IGRIP for *World Model* display and animation, and also runs associated data communication processes. A Sun computer was also interfaced to the control environment for display of various non-geometric aspects of the *World Model*. Interfaced to the SGI are multiple Motorola 68030 single-board microprocessors resident in VME backplanes, which provide real-time, sensor-based control. The RIPE interface software translates the high-level commands from the SGI into sequences of low-level commands understood by the controllers and sensors. The robot controller and sensors are interfaced to the computing environment by the Sandia developed Intelligent Robot Operating Environment (IROE).[10] IROE is a real-time, multitasking operating environment built upon the VxWorks operating system (Wind River Systems, Inc.), and was specifically developed for the communication demands of real-time sensor directed-control of intelligent robot systems.

The graphics environment is an ideal environment in which to off-line program the robotic system. To minimize system downtime or to limit operator programming time on the system it is desirable to pre-plan the intended motions of the robotics system as much as possible. Accurate 3-D geometric models of the robot, its end effectors, and the fixtures and workpieces in the environment allows the robot programmer to carefully plan robot trajectories. Since the robot motions are simulated using accurate kinematic models, the pre-planned motions faithfully reproduce the actual motions of the robot. In addition, task execution time can be determined and analyzed for system bottlenecks.

Each robot typically has its own high-level programming language. The code to program the robot is written much like any other code, with the exception that the positions the robot is to move to are manually taught to the system. The integration of the graphical interface with an input device allows programming of the motions of the robotic system without having to write code. This can greatly speed the programming process and decrease the programming skill level required of the operator. Programming is thus realized by the operator causing motions of the simulated robot, the computer system remembering the commanded motions, replaying and displaying the commanded motions to the operator, and then with operator acquiescence, downloading the programmed motions to the robot controller and causing the motions to be executed.

The system software is written in such a way that any input device (*e.g.*, teach pendant, spaceball, force-reflecting master) can be used. The control commands from the input device are sent directly to the graphical system. The control system determines how these commands are to be interpreted, (*e.g.*, the movement of individual joints, or end point control based on a tool frame). The result of these commands are displayed real-time in the graphics environment. The motion information is combined with the collision detection capability contained in the *World Model*. Thus the operator can be alerted to the close approach of the robot to known objects in the environment.

The very nature of the graphics model allows for the ability to detect collisions between modeled objects. This ability is very important for the safe programming of robotics systems. It is undesirable for a robot to have an uncontrolled collision with objects in its environment. Such interactions are typically conducted at slow speeds, with force sensors measuring and controlling the forces of interaction. While an operator's attention is generally focused on the end of the robot and on the task at hand, another part of the robot, such as the elbow, may collide with objects in the environment. The collision detection

capability of the graphics environment detects and warns of all such impending collisions. In addition, both the parts selected for collision detection and the warning distance are user- selectable. This ability allows the task planner to select the objects of importance and to dynamically alter the warning levels based on the task at hand. This also allows for different collision detection capabilities depending on the skill of the operator.

6.1 System Operations

We have presently employed this system in two laboratory-scale systems and one full-scale demonstration for critical feature testing. In each case, the graphical control environment is used as the programming and control interface for GISC to produce a faster, safer, and more economical system. The first application of this technology was a laboratory demonstration relating to the remediation of nuclear waste buried in underground storage tanks.[11] The second application of this technology was the safe tele-operated inspection of a nuclear waste transportation cask.[12] The full- scale demonstration involved three robots working together to map a mock underground storage tank and then remove the simulated hazardous material.[13]

The graphical control environment described has been implemented and demonstrated in a Sandia laboratory test facility designed to demonstrate the robotic characterization and remediation of hazardous waste stored in underground tanks. The foundation *World Model* contains 3-D geometric models of the system generated from construction drawings. The system and its graphical representation can be seen in Fig. 3. The test work environment consisted of a gantry robot (Cimcorp XR6100) and a 2.4 by 2.1m rectangular tank filled to a depth of 0.6m with moist sand to represent the waste. The test tank contains both buried objects and pipes, as well as structures protruding above the surface of the waste to represent the types of obstacles that would be encountered during characterization and clean-up of actual storage tanks. The waste surface and some obstacles were deliberately not modeled to test the control system's ability to detect and map unknown obstacles. The *World Model* also contained models of the robot's kinematics and motion limitations and heuristics defining tasks such as mapping and waste removal. The initial robot motions were defined by operator, tested for safety and potential collisions within the *World Model* and executed. As motion occurred the robot model was driven by information received from the robot's encoders. As the sensor systems detected new information about the environment, such as waste surface profiles the data was communicated to the *World Model* and graphically displayed.

The second application of the graphics control environment is the inspection of nuclear waste shipping containers. This system also made use of the Cimcorp gantry robot in conjunction with a quarter-scale mockup of a waste transportation storage cask. Through use of the graphical interface, the operator can direct inspection tasks around the container. The operator can pick up tools and direct inspection tasks with the assurance that collisions will be prevented by the use of the graphics control environment. This capability will be essential for safety inspections and during emergency operations. The geometric *World Model* and the graphics interface to the operator are used to graphically program the robot system and to verify safe operation. For example, prior to the start of a series of programmed operations, the computer compares the desired robot trajectories with the geometric knowledge contained in the *World Model*. Any unsafe trajectory (*e.g.*, a collision between the robot and a model feature) is detected and reported to the operator via the graphics interface prior to robot motion. The operator can then modify the proposed robot trajectory or program the robot by simply manipulating the robot image in the graphic interface. These modifications to the robot trajectories are then verified for safe operation by the *World Model*. The computer system automatically reprograms the robot's movements to include the operator's commands. Only collision-free robot trajectories are transmitted to the robot.

The full-scale demonstration encompassed and enhanced the capabilities developed in the laboratory demonstrations. The expanded capabilities included interaction and control of three different robot systems, increasing the system's ability to handle a variety of work environments, and a number of different types of tools and sensor types. The system consisted of: a RedZone RTI robot used for tank wall and weld inspection; a Schilling Titan 7F robot used for fine manipulation tasks mounted on the end of a 28-foot-long Spar hydraulic arm used for gross positioning. A drawing of the system can be seen in Fig. 4. Also the graphical representation can be seen in Fig. 5. The tooling included an ultrasonic mapping tool, a hydraulic cutter, and a small pneumatic chipper.

The increased number of robots and tools tested the system's ability to program and communicate to different systems in the same language and using the same programming tools. The control structure and the command flow from the reasoning system to the real-time controller allowed seamless communication and control to each device. It also demonstrated the ability to perturb

commanded motion from updated real-time information. As in the laboratory demonstrations, surface information was mapped by the sensors and included in the *World Model*. Based upon this mapped information, ideal task trajectories are generated. When these trajectories are executed the sensor system perturbs the motion to maintain a desired height above the surface, for example. The results of the perturbed motions are fed back into the graphics system informing the operator of the changes made to the original desired trajectory.

In addition, the use of virtual camera views for operator feedback was implemented. The operator's view of the graphics representation of the *World Model* was adjustable. Through use of a set of dials the operator could modify the scale, translation and rotation of the view or views as he/she desired. This ability allowed for more detailed inspection of those areas of the model which were of particular interest. It should be noted that full collision detection of the entire model environment was always available regardless of what was displayed on the screen. In the particular case of removing and replacing tools from the tool rack, the virtual camera view was clearer and more valuable than the actual camera view which was cluttered and often obscured the desired information.

7. RESULTS & CONCLUSIONS

The use of a graphical control environment for robotic systems can accelerate tasks such as the removal of waste stored in underground storage tanks. Advanced 3-D geometric modeling concepts can allow robot motion planning and thus, facilitate programming the system without manual code generation. This greatly reduces the requirements for detailed, step-by-step programming by skilled robot programmers. The geometric model can interpret operator manipulations of the graphical representation to automatically program the robot to respond in the manner desired by the operator. In addition, the *World Model* can also be used to validate operator commands to the robot system to ensure safe operation during manual control. Graphical display of the results of the operator's robot commands can provide operators with perspectives not available from direct video viewing commonly used in the tele-operation of remote systems. This increases system safety by warning of impending collisions, and is especially important for impending collisions away from the robot end effector where the operator's attention is frequently focused. If an operator's command could result in a collision, the control system, with reference to the *World Model*, can prevent execution of the command by the robot and communicate the source of the problem to the operator through the graphics interface.

8. ACKNOWLEDGMENT

Much of the work described in this paper was performed while the author was at Sandia National Laboratories. The author would like to acknowledge the work of William Drotning, Peter Boissiere, William Davidson, Michael Griesmeyer, Sig Thunborg, and Ray Harrigan in developing the software and hardware interfaces that made Graphical Model based control possible.

9. REFERENCES

1. Hopcroft, J., (1986). Impact of Robotics on Computer Science. Communications of the ACM, Vol. 29, #6, 486-498.
2. Martin, H. and Kuban, D., (1985). Teleoperated Robotics in Hostile Environments, Dearborn: Robotics International of the Society of Mechanical Engineers.
3. Boissiere, P., and Harrigan, R., (1989). An Alternative Control Structure for Telerobotics. Proceedings of the 1989 NASA Conference on Space Tele-robotics, vol 5, p. 141.
4. Brooks, M., and Lillekjendlie, B., (1989). A Graphical Programming Environment for a Grinding Robot, Proceedings of 20th International Symposium on Industrial Robots, Tokyo, Japan.
5. Tarnoff, N., Jacoff, A., and Lumia, R., (1992). Graphical Simulation for Sensor Based Robot Programming, accepted for publication in Journal of Intelligent and Robot Systems
6. Tarnoff, N., and Lumia, R., (1988). The Role of Off-line Robot Programming in Hierarchical Control, Second International Symposium on Robotics and Manufacturing Research, Education and Application, ISRAM, Albuquerque, NM.
7. Deneb Robotics, (1992). IGRIP, Auburn Hills, MI.
8. Harrigan, R., (1992). Generic Intelligent System Controller Review, in Progress
9. Miller, D., and Lennox, C., (1990). An Object-Oriented Environment for Robot System Architectures, Proceedings of the 1990 IEEE International Conference on Robotics and Automation, vol. 1, p. 352.
10. Davidson, W., (1992). IROE, An Intelligent Robot Operating Environment, to be published.
11. Christensen, B., Drotning, W., and Thunborg, S., (1991). Model Based, Sensor Directed Remediation of Underground Storage Tanks, Proceedings of the 1991 IEEE International Conference on Robotics and Automation.
12. Drotning, W., and Bennett, P., (1992). Graphical Models for Simulation and Control of Robotics Systems for Waste Handling, to be presented at the 3rd International High-Level Radioactive Waste Management Conference
13. Christensen, B., Griebenow, B., and Burks, B., (1991). Graphic Model-Based Control of Robotic Systems for Waste Remediation, Transactions of the American Nuclear Society, 64, 609.

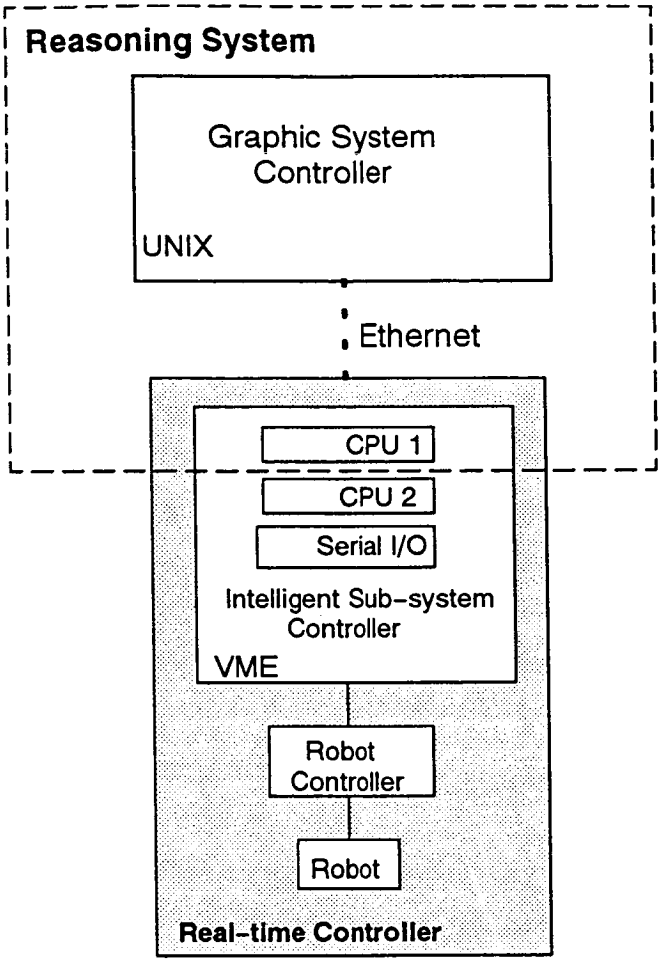


Fig. 1 System Architecture

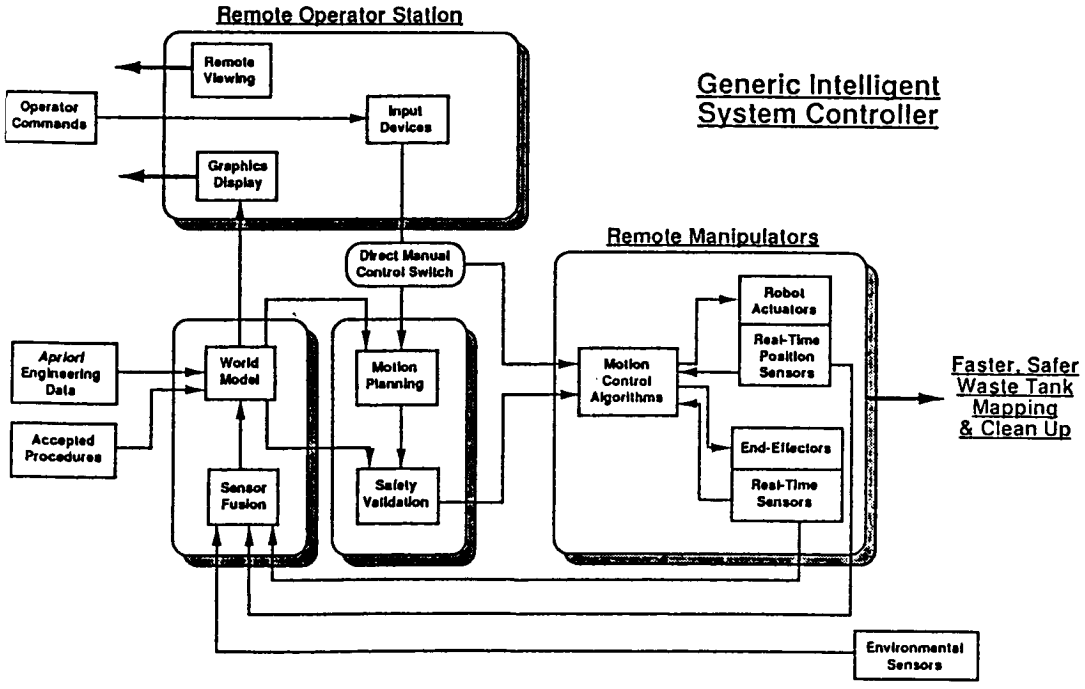


Fig. 2 Distributed GISC Environment

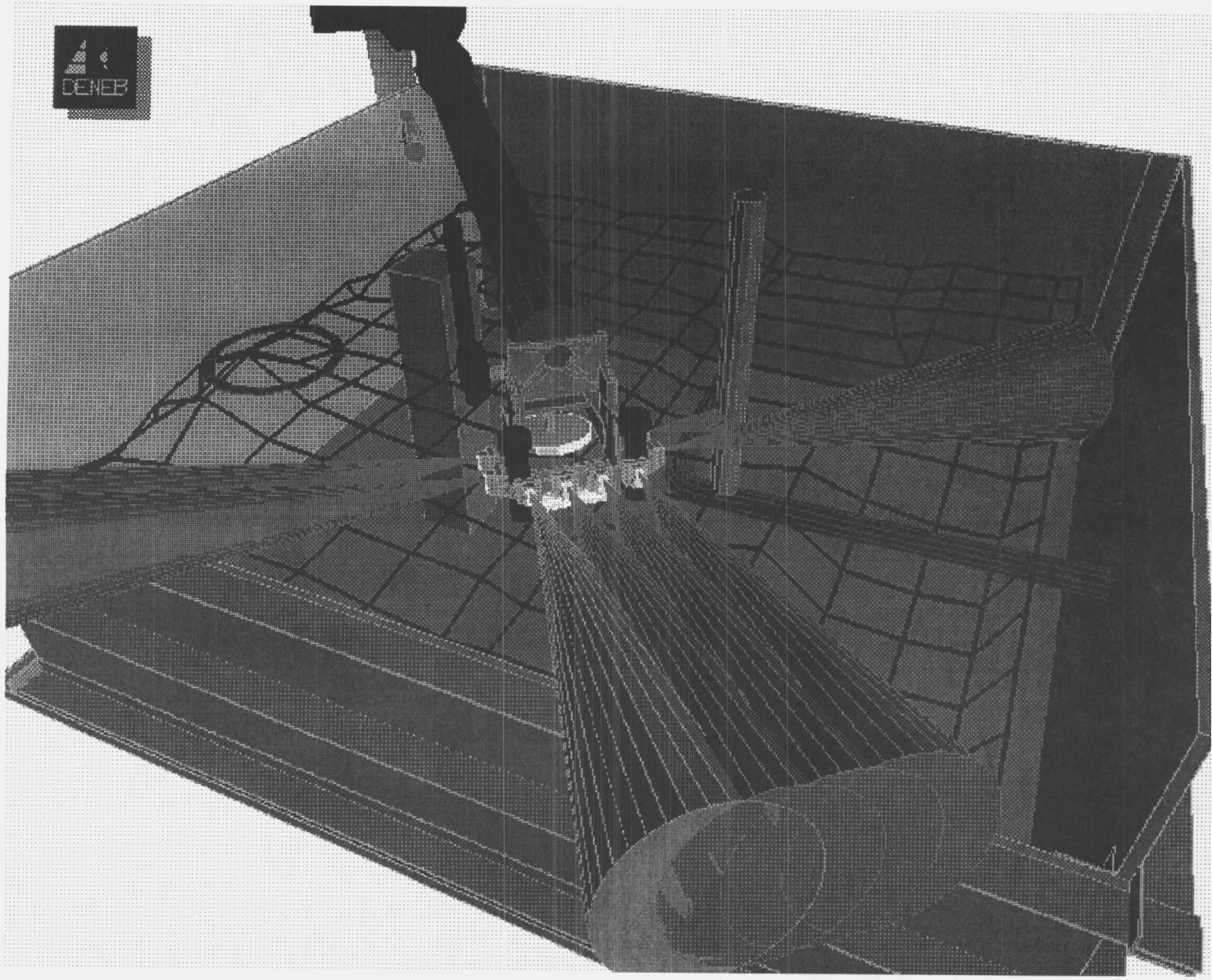


Fig. 3 Graphical Representation of Waste Remediation Test Bed

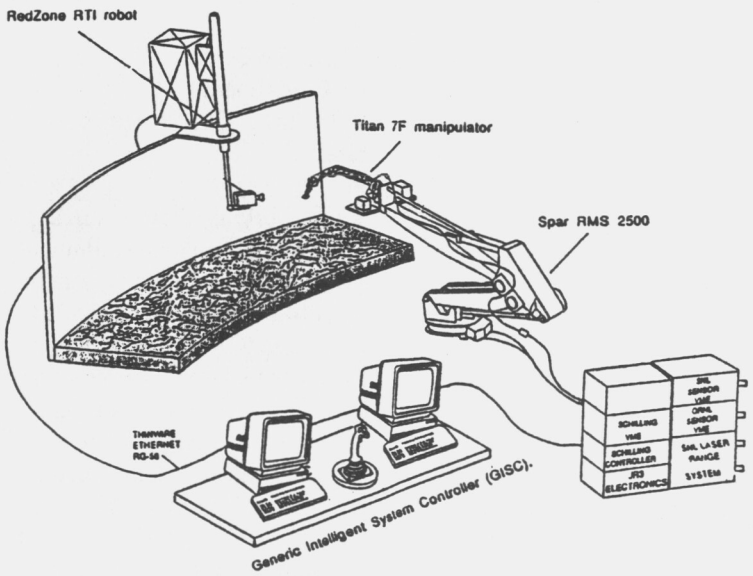


Fig. 4 Full-Scale Waste Remediation System

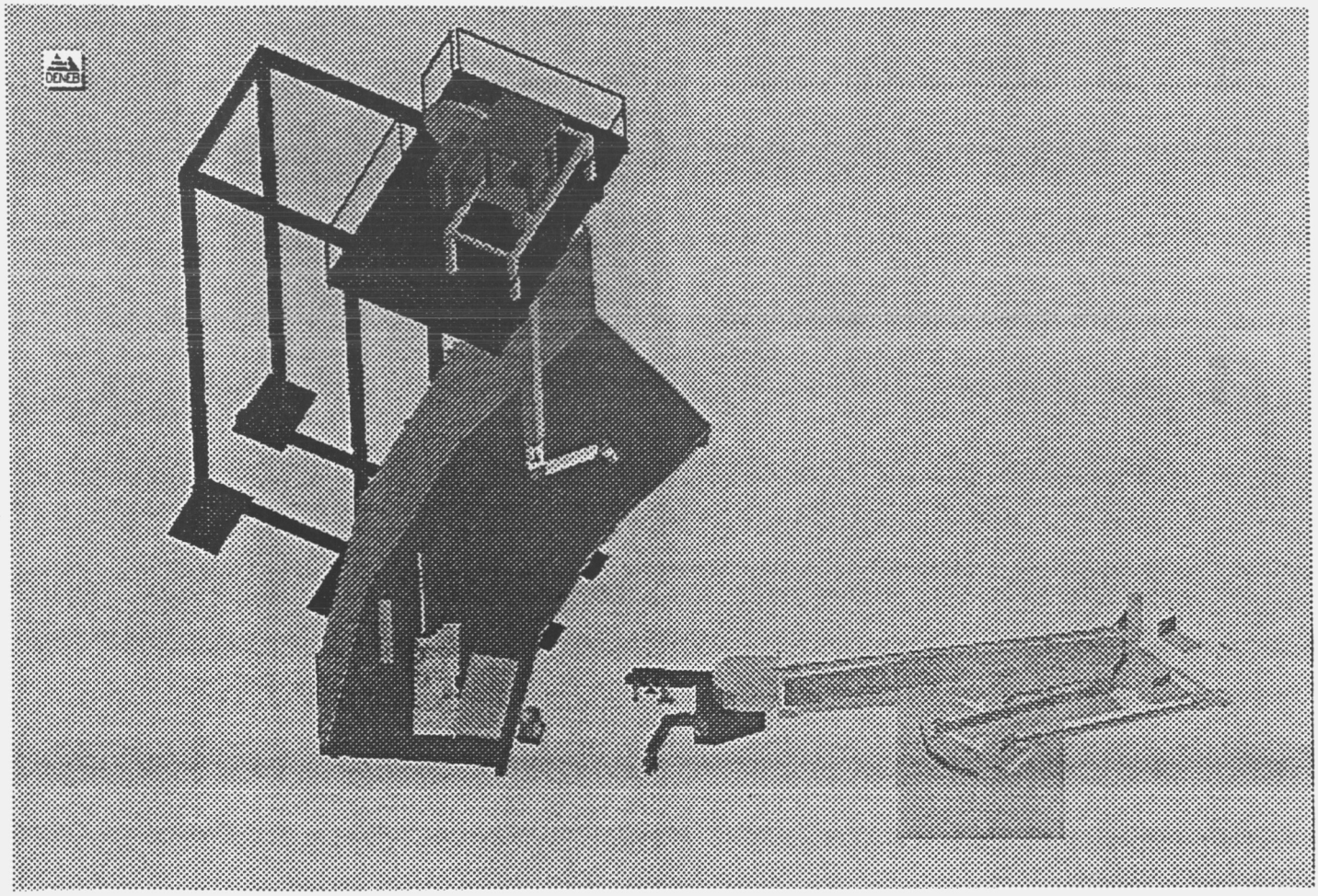


Fig . 5 Graphical Representation of Full-Scale Waste Tank Remediation System

DESIGNING THE NEXT GENERATION OF ROBOTIC CONTROLLERS

David G. Goldstein, Ph.D.
Computer Science Department
North Carolina Agricultural & Technical State University
Greensboro, NC

Abstract

This paper describes the scenario-based, object-oriented approach used to specify the software architecture of the next generation of robotic controllers. We also discuss how we intend to implement a version of the controller via a multi-agent approach. We also describe our real-time, fault-tolerant, cooperative reasoning tools that we intend to use to facilitate developing the implementation of the controller. We also describe how we intend to interface existing applications and controller components to the tools so that they interact via objects detailed in the controller specification.

Introduction

Problem

The manufacturing industry of the United States has become increasingly less effective as other nations pour money into research. To regain world technological pre-eminance, the Advanced Research Project Agency and the National Center for Manufacturing Sciences have launched a number of programs. One of these, the Next Generation Controller program, attempts to develop a standard for robotic controllers for the post-1995 time frame. This paper will discuss the approach taken during our working for the program while staying within ethical boundaries concerning this crucial research program. The interested, authorized reader can obtain the actual specification document can be obtained from the National Center for Manufacturing Sciences.[1]

Most of the robotic controllers used in America today reflect programming concepts that are decades old. Change is required to remain competitive. However, manufacturers will not invest in equipment unless the return on investment promises to far exceed any risks. Therefore, the characteristics of any controller proposed must satisfy two different general goals: risk reduction and performance.

Robotic controllers must ensure several key items to facilitate risk reduction. First, current manufacturing practice must be supported; manufacturers are loath to shut down a working factory on a promise of efficiency. Second, current equipment must be supported. Finally, numerous existing applications and utilities must be supported. While each of these items will be discussed later in this paper, open systems largely address these concerns.

Innovative robotic controllers must also provide vast performance improvements for acceptance, where "performance" encompasses several aspects. Obviously, controllers almost always have real-time deadlines to meet and so should afford accurate results via efficient computational processes. Controllers should also afford previously unavailable capabilities, better user-interfaces, and promises of more efficient programming. Since product acceptance eventually depends upon economics, either flexibility (expanded product-line or higher quality product) or lower life cycle cost (through lower product development costs) must be offered.

Loss of market share and un-transferred technical innovations prompted the programs adoption of both risk reduction and increased performance. Current practice in the robotic control industry has hardly changed in decades (with the exception of a few, very well financed areas). The programs goals attempted to afford advanced features (such as art-to-part manufacturing) while facilitating a smooth transition from existing equipment and support software to the next generation of machinery.

Program History

This paper will try to explain the approach that was eventually, successfully used in developing a specification for the controller: scenario-based object-oriented analysis. This approach will be the central focus of the paper because traditional approaches failed dismally in attempting to gain enough support for developing a standard. Another key aspect of developing the standard was "cleaning the kitchen"; if too many cooks don't make good broth, then using fewer cooks might help. It did.

Copyright c 1994 by David G. Goldstein. Published by the American Institute of Aeronautics and Astronautics, Inc. with Permission.

Organization of Paper

We first describe the scope of the controller's specification to provide context for the rest of the paper. We then describe the scenario-based methodology and how it can be employed to specify a controller. We then provide a description of the contents of such a specification. We finally analyze the effectiveness of our approach and draw some conclusions concerning the utility of the specification and the effectiveness of the proposed controller.

Scope of Controller Specification

Developing a national standard for robotic controllers necessitates employing a wide brush; painting a picture of robotic use and manufacturing needs in the near future requires comprehensive coverage of the domain while affording a great deal of flexibility. Comprehensive coverage is required to examine manufacturing from the level of controlling motors in actuators to analyzing throughput of factory lines. We try to specify interfaces to almost any type of information that a user of the system might want to analyze or modify.

Comprehensive coverage is also reflected in the very-varied concerns examined by the specification. Physical elements, such as sensors, effectors, payloads, conveyors, tools, and users are considered. Abstract physical elements, such as envelopes, schedules and enterprise expectations should also be considered. Abstract elements, such as those for configuration, are by far the most difficult, essential elements to include; different implementations employ different strategies for configuration, different kinds of elements requiring configuration, and different granularities of configuration. Given that the controller should address needs as diverse as composite baking, precision material removal, and many-axis (> 100) assembly, facilitating comprehensive coverage often required multiple, combinable representations.

Comprehensive coverage also includes covering current practice: in terms of specifying interfaces to existing machinery, programming techniques, and support software. Hence, interfaces to facilitate interacting with programs for solenoids and C++ programs were both supported.

The controller specification also reflected a great deal of flexibility. Interoperability and plug-replaceability are essential in new robotic architectures destined for commercialization. Such flexibility is facilitated by employing existing standards and an open, published architecture. With respect to this endeavor, the obvious standards to employ were EXPRESS and the Product Data Exchange Specification.[2,3]

The Product Data Exchange Specification (PDES) attempts to provide a standard mechanism for describing virtually any object that might be manufactured as a product. Hence, the numerous volumes of the specification are appropriate for addressing a variety of fields. The standard is hierarchical, building upon very primitive concepts such as Cartesian coordinates and measurement units to eventually describe features such as pockets and items such as resistors.

PDES was written in the specification language EXPRESS. EXPRESS is an ISO standard (but currently undergoing revision for its next version). EXPRESS is well-suited for describing object-oriented concepts (see Object-Oriented Methodology), since it supports both inheritance and abstract data types.

Inheritance is the notion that items described at a higher level in a hierarchy are subsumed in the structure of objects placed lower in a hierarchy (e.g., all mammals have mammary glands). Abstract data types allow new representations for new items (e.g., a car can be represented as a data type and used in representing a fleet of cars).

An example of EXPRESS code is provided in Figure 1. The first entity (object described) is a representation for a sine-wave. Such a type of line should inherit the characteristics of general Line's and is more abstract than lines described as Cartesian_sine_line and Spherical_sine_line.

A sinusoidal line has several attributes, to describe the phase, amplitude and compression factor of the wave. EXPRESS also facilitates constraining the attributes of an object. This example uses the functions all_in_0_to_2pi and periodic_phenomenon to describe relationships among values of attributes that must be present in valid objects.

The specification of the sine-wave is used in the second entity to describe a series of sinusoidal lines (such as might be generated by a

Fourier transform). The example allows for any positive number of sine curves to be combined to describe a Sine_series_line. The constraint expressed here states that there must be an offset for each curve specified.

Much of the work in the United States with respect to both EXPRESS and PDES stems from the National Institute for Standards and Technologies (NIST). The ISO acceptance of EXPRESS as a standard has prompted a large number of tool vendors to also support EXPRESS. Similarly, a great deal of funded research employs PDES to encourage its acceptance. NIST also has developed a public domain toolkit for manipulating EXPRESS models to facilitate working with PDES.

```
ENTITY Sine_line
  SUBTYPE OF (Line)
  SUPERTYPE OF (ONEOF(Cartesian_sine_line, Spherical_sine_line)) ;
    (*
    An individual line described in terms of
    Ampliture*sin(PeriodCompression*Angle+Phase)
    *)
     Compression_coefficient : REAL;
     Phase : REAL;
     Ampliture : REAL;
    WHERE
    all_in_0_to_2pi(Profile_element.Phase);
    periodic_phenomenon(Sine_line);
     END_ENTITY;

ENTITY Sine_series_line;
    (*
    Represents a function as a summation of sines of the form:
    f(w) = C1*sin(A1w+B1) + C2*sin(A2w+B2) + C3*sin(A3w+B3) + ....
    Offsets (t0, t1, ...) are also provided so that relative calculations can be performed, as in
    f(w) = C1*sin(A1(w-t0)+B1) + C2*sin(A2(w-t0)+B2) ...
    *)
    Offset: LIST[1:#] OF REAL;
    Component: LIST[1:#] OF REAL;
   WHERE
    SIZEOF(Offset) = SIZEOF(Component);
END_ENTITY;
```

Figure 1: Sample EXPRESS Model

Object-Oriented Methodology

Our goal in developing the robotic controller specification was to provide a set of standard interfaces by which various applications and equipment could communicate. We provided this interface by describing the set of objects that could be transmitted among applications and support software. We also specified the constraints placed upon these objects to ensure their validity. We also described some minimal performance requirements required of various classes of applications when generating and manipulating these standard objects.

The object-oriented methodology employed and developed novel concepts in Object-Oriented Analysis (OOA) and Design (OOD). Because public domain tools support automated translation of EXPRESS code into C++ (arguably the most popular language considered to be "object oriented"), Object-Oriented Programming (OOP) is also supported.

OOA strives to discern the essential objects for describing a domain. OOD attempts to organize and describe these objects, their behavior, and their interactions. We employed a specific object-oriented technique, Scenario-based Object-Oriented Analysis and Design, to derive the controller architecture. These techniques are particularly good at clearly expressing concepts in a domain and at providing an audit trail to the source of the

original object. To improve the clarity and utility of the our work, we employed Computer-Assisted Software Engineering (CASE) tools to express the products of our analysis and design.

Terminology and Procedures

Object-Oriented Software Engineering is a relatively new approach for developing software. The approach treats instances of data types as *objects*. The data types themselves are typically called *classes*. Walt Disney's favorite car, "Herbie", would be considered an object of the class "car".

Classes have attributes describing characteristics of objects. These attributes are assigned values to reflect the characteristics of particular objects. Hence, the color attribute of the class "car" with respect to the object "Herbie" might have the value "white".

Object-oriented techniques provide numerous benefits, a description of which would be beyond the scope of this paper. However, two of these advantages previously mentioned (abstract data types and inheritance) facilitate developing hierarchies of objects.

The *Composition Hierarchy* pictured in Figure 2 was described via EXPRESS LIST's in Figure 1; composition hierarchies express how multiple objects can be combined to describe more complex objects. Machines are excellent examples of composition hierarchies: a complex machine including tools, spindles, links, etc., can be succinctly expressed in a composition hierarchy.

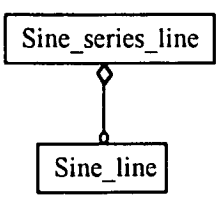


Figure 2: Composition Hierarchy

The *Generalization Hierarchy* expressed in Figure 3 was described via the SUPERTYPE and SUBTYPE relationships as part of the EXPRESS code in Figure 1. Generalization Hierarchies facilitate less abstract classes inheriting attributes from more abstract classes. A clearer example might be machines: Abbrassive_waterjet would be less abstract than Material_removal_machine that would be less abstract than Making_machine, which would be less abstract than Machine. Abbrassive_waterjet's inherit attributed from Material_removal_machine that inherit attributes from Making_machines that inherit from Machines.

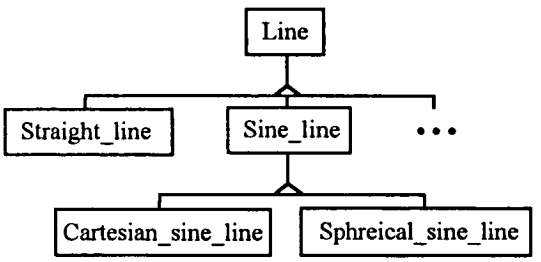


Figure 3: Generalization Hierarchy

Scenario-based Techniques

We employ *scenarios* to ascertain the various objects, their behavior, and their interactions in a given domain. Scenarios are timelines of events with respect to one or more objects. One can imagine a scenario as a movie depicting the existence of an object at some level of granularity.

Determining the level of granularity for examining an object is essential and often non-trivial. A machine can be described by its shape, or the shapes of its components, or by the shape of pieces of the components (such as screws on a jig).

Granularity is complicated by the fact that many aspects of an object might require descriptions in different measurement units and at multiple levels of abstraction. For example, if a planning application is going to reason about the behavior of a machine, it must know the machines' capabilities. The planner must know what the machine can do and with what precision. To ascertain the interactions among objects affecting the machine's precision, we may have to examine both a more precise granularity of composed objects (e.g., interactions of surfaces of the jig and tool) and a more precise granularity of time (e.g., microseconds as opposed to seconds). Hence, we use scenarios describing the same objects, but encompassing different time granularities.

Scenario-based analysis is particularly effective because it directly maps elements in the domain to models in the interface. Scenario-based analysis builds customer confidence

because he has a better concept of the mechanisms underlying any "black boxes". Scenario-based analysis also facilitates traceability; because classes originate from particular scenarios, questions that a customer might have can quickly and efficiently be addressed. We tend to involve the customers as much as possible in the analysis phase, hopefully obtaining the actual scenarios from them via interviewing techniques.

Implementing a Controller

We hope to implement a version of the specification using a multi-agent approach. Recent literature in artificial intelligence suggests that collections of simple agents are much easier to control than large, monolithic programs.

We intend to treat applications and portions of the implemented controller as intelligent agents (Figure 4). Many of these agents, e.g., planners, will be represented as knowledge-based applications. Other applications, e.g., machine executives, will be embedded in wrappers to communicate standard objects from the specification.

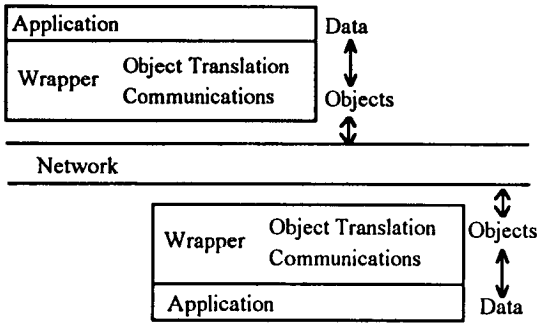


Figure 4: Controller Components as Agents

Crucial to our implementation of a controller will be the Distributed Artificial Intelligence Toolkit (DAIT).[4,5] DAIT provides distributed knowledge-based processing while affording transparent processor fault-tolerance. DAIT also includes predicates to facilitate real-time control. DAIT is based-upon NASA's C Language Integrated Production System.[6] DAIT includes tools for metering, configuration, interprocess communication, and user-interfaces. DAIT supports forward-chaining reasoning, procedural programming, functional programming, object-oriented programming, and deductive database queries. We expect to implement both Fuzzy inferencing and backward-chaining in the near future.

Conclusions

Object-oriented analysis and design provide an attractive mechanism for examining the domain of robotic control. Scenario-based software engineering techniques can often be used to achieve consensus among multiple customers concerning requirements. The Next Generation Controller program successfully used these techniques in developing a specification for standard robotic controllers in the U.S.

We hope to soon implement a version of the controller by employing multi-agent techniques. We will use internally developed tools specifically designed for cooperative knowledge-based processing in this endeavor. By embedding existing software in wrappers communicating objects found in the specification, we will facilitate interoperability and interchangeability of various components of our controller.

The next generation of robotic controllers must be innovative enough to support avant-garde research concepts such as art-to-part manufacturing. However, this specification of these new controllers will also have to address the needs of supporting current hardware and software. We feel that the specification adequately addresses these concerns. We also hope to soon realize a controller demonstrating interoperability and interchangeability of components while offering a very high degree of functionality.

References

[1] NATIONAL CENTER FOR MANUFACTURING SCIENCES,

[2] INTERNATIONAL STANDARDS ORGANIZATION, 'Product Data Exchange Specification First Working Draft', ISO TC184/ SC4/WG1, Document N284, 1988.

[3] SPIBY, P.: 'EXPRESS Language Reference Manual', Document Number N14, International Standards Organization, April, 1991.

[4] GOLDSTEIN, D.: 'The Distributed Artificial Intelligence Toolkit', *AI Expert* (Miller-Freeman Publishing), San Francisco, January 1994.

[5] GOLDSTEIN, D.: 'A Fault-tolerant Architecture for Distributed, Heterogeneous, Deductive Knowledge-based Applications', Ph.D. Dissertation, University of Texas at Arlington, Arlington, USA, August 1992.

[6] GIARRANTANO, J.: 'CLIPS User's Guide' (NASA/JSC), Houston, TX, 1991

Acknowledgments

Aspect of this work were inspired while under contract on the Next Generation Controller program. The author has continued to investigate concepts presented here as part of North Carolina Agricultural & Technical State University's Manufacturing Initiative (internally funded) and Generic Object-Orioented Software Engineering laboratory (partially funded by the Army and IBM).

AN END USER'S WISHLIST

BY CHARLES W. WARD

It was 1983. I was standing in a Sears store at 5:00 in the morning knowing there had to be a better way. We had no one to clean the 180,000 sq/ft store and the doors were to open at 9:00 a.m. sharp. Considering the average cleaner works at a rate of 3,750 sq/ft per hour I knew I had a problem. There was no way to delay the opening by asking management to hold off their customers until my crew showed up. The show had to go on! So I rounded up who I could on the phone and we limped through the morning, just barely finishing as the crowds poured through the doors. This was not just a bad day. It was a normal day. That was the year I began my search for a mobile robot that could clean.

This picture of our cleaning operation is not unique. It happens everywhere, in every company, every day and in every city in this country. Absenteeism is a routine daily occurrence for those of us in the cleaning industry. But fortunately, there are those employees who do come to work every day. Employees we know we can depend on. But, these are the employees everyone wants. They are in demand. Therefore, they tend to leave us for more challenges, lucrative offers and better careers with more advancement and educational opportunities. In addition to the problem of employee absenteeism we also have to manage massive employee turnover.

I believe I can summarize the general problems in the cleaning industry in the following way: how to fill the 800,000 new jobs that will be created in our industry by the year 2000; how to reduce turnover from an industry average of 400% a year, how to reduce absenteeism and increase morale, and how to attract competent labor. Those companies that are experience to these problems are looking at mobile robots as a potential solution.

My purpose today is to share with you some of our ideas about what applications might be roboticized and what factors are important to us as end users in the cleaning industry.

A close friend and associate in the mobile robotics business once said to us, "The ideal place to apply a robot is where a human is working like one." Using this advice as a guide, we can identify many opportunities for automation. This is not to say that cleaners don't have feelings or emotions, but rather the majority of their work is routine, mundane and repetitive.

Repetitive tasks that we have considered automating are floor scrubbing, floor sweeping, floor polishing, vacuuming, trash removal and bathroom cleaning. Of course, other tasks like dusting and waxing floors could eventually be automated as technology

Copyright c 1994 by Charles W. Ward. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

improves. But our opinion is that when you decide what task you will automate you should consider the simplicity of the task and engineering capability as well as return on investment.

Some companies we have known have made the mistake of assuming that once they have isolated some tasks to roboticize, have run some numbers, raised money for development and adapted their robotics technology to a cleaning machine they would be ready for sales. But, unfortunately, it is not that easy! Our opinion is that before developing a robot one must keep three major things in mind: the environment that the robot will work in, the market the robot will be sold to and the workers it will interact with.

First, I would like to discuss the operating environment. The diversity of building design and the changing environment within a building make it necessary for a robot to be built that is flexible and easily adaptable. In the cleaning environment things change. People move offices, walls and desks. They often leave huge obstructions in the hallway at night. The reality is that no one wants to move that big box in the hallway. And people will not want to go and rescue a robot when they have their work to do.

We have experienced two schools of thought that attempt to address the navigation of a complex environment. The first is to teach the robot everything about the building and provide it a map for navigation. Even though we believe this is still the most efficient way to navigate we certainly can't have a robot that takes months to program.

The second navigational strategy is preprogramming or self teaching. We have had some experience with these types of robots and while they help to address the simplicity issue they fall short in the performance area. Again, the reason is that each environment is uniquely its own and the decision about how to navigate these changing environments are being left up to a machine! I know if I still get lost in buildings a robot is likely to do the same! After some experience we believe that preprogrammed robots major drawbacks are inconsistency and they require too much human intervention time to operate.

The robots we operate must be consistent performers. Consistency is critical to establishing a corporate image of quality. This is especially true in the cleaning industry. The industrial robots that weld, spray paint and assemble the quality cars we see today are consistent and reliable. Mobile robots that clean must be just as dependable and reliable. We cannot build our operation or reputation around unreliable people. The same is true for a robot. Please bear in mind, we'd rather have a robot clean 1000 square feet well every day than a robot that sometimes cleaned 10,000 square feet and sometimes didn't.

The second consideration to make will be the market to which the robot will be targeted. Although not a glamorous business, cleaning is a stable, secure and large industry. Currently the

industry is 50 billion dollars a year here in the U.S. alone and it is expected to grow to 80 billion by the year 2000. But because of the enormous opportunity and the low barriers to entry the industry is flooded with competition. This competition is diverse in the way it looks at its customers, operations and its people. In the cleaning industry you have every kind of businessperson. From the mom and pop cleaning companies who operate out of the trunk of the car to huge multinational companies who do a billion dollars in cleaning a year. The mentality is significantly different. It is critical to know who will buy your unit and adapt the sales pitch accordingly. I believe it will be most difficult to build a robot that can be used throughout the entire industry. We believe success depends on finding a niche to begin.

Pricing robots for the target market is also a difficult issue. As I said before, each building is unique and its operation expenses vary accordingly. Some buildings contract their cleaning and some maintain their own staff. Some buildings have all carpet and some all tile, but most a combination of both. We believe that in order to effectively market a robot individual studies of each location should be done. To get to a real return on investment number real data about labor rates, production rates and total cleaning cost are mandatory because these figures can vary from building to building. For example. general office cleaning labor rates in central Virginia are at an average of $5.00 per hour. But in industrial plants around the same region they are at an average of $11.00.

We have included a real analysis that my company did for a customer in Virginia that is considering automation. This customer desired a one year or less payback period and wanted to roboticize his vacuuming process. We make the assumption that the robot will work primarily in the wide open spaces within the building where there is little obstruction. In this example, where we have shown a 7 month payback on a $22,000 investment for two robots, it is important to note that there is no human intervention time required. The more human intervention required the longer the return on investment will be. (See Exhibit A).

In addition to adaptability, consistent performance and target market the last consideration and the most important factor is people. Who will the robot interact with? Who will run it? Who will service it? And who will reeducate an industry that has not had any major innovation since the vacuum cleaner? It is the biggest mistake of all to assume once you have produced an efficient and reliable machine that the industry will be beating down your door to buy one. There will be an enormous educational problem for our industry once technology has begun to be integrated.

Producers of mobile robots must be prepared for that educational challenge. They must be prepared to participate in reshaping, rethinking and supporting an industry that is inevitably facing major changes in the next century. Designers and developers

number one barrier to market acceptance is this human factor. It is critical to develop robots that serve people, not ones that people need to serve. Those robots thought of as "user friendly", however vague the term, will be accepted quicker and stand the greatest chance of success. By "friendly" I mean things like; ease of programming, self diagnostics, manuals written in simple english, "service"-minded technicians, the absence of hidden costs and reliability.

I have listed some detailed operational specifications I think are important to consider when developing a robot for the cleaning industry. (Please see "The WishList" Exhibit B). But ultimately the success of a mobile robot in cleaning will be dependent on the same thing as it is for any product: how the company who sold the robots supported and serviced the people who are using it.

EXHIBIT A "CASE STUDY"

KNOWN:

750,000 Cleanable square feet (All areas)

375,000 square feet carpeted

125,000 square feet open and unobstructed area ideal for robot application

Cleaning done between 6:00 p.m. and 10:00 p.m. Monday through Friday or 260 days per year

Cleaning done using part time labor

Human can vacuum 6000 square feet per hour

Wage per hour is $6.00 including taxes and insurance

ASSUMPTIONS:

Robot per unit cost: $11,000

Robot annual maintenance: $500 per unit

Robot production rate: 9000 square feet per hour

Robot to clean 365 days per year

Robot to work 6:00 p.m. to 1:30 a.m. daily

Expected life of robot: 8 years

ANALYSIS

	HUMAN	**ROBOT**
Daily hours needed to vacuum	21	14
Days worked each year	260	365
Annual hours needed to vacuum	5,417	5,069
Annual labor cost	$32,500	_____
Total robot investment (2 units)	_____	$22,000

PAYBACK PERIOD:

$22,000 (INVESTMENT)

Annual labor - Annual maintenance

$32,500 - $1,000

= .7 Years (8 months)

Annual savings = $31,500

Over the 8 year life of the robot: $230,000+

RETURN ON INVESTMENT

Annual labor - Annual maintenance

$32,500 - $1,000

$22,000 Investment

= 143% Return on Investment

EXHIBIT B: "THE WISHLIST"

---

1 Robot should be easy to program

2 Robot should be self charging

3 Robot needs to be truly "Autonomous" or able to run without any human intervention

4 Robot needs to be able to navigate multiple hallways

5 Run time should be a minimum of 4 hours

6 Cleaning path should be as wide as possible but robot should be able to go through standard doorways

7 Robot should be able to call elevators and move from floor to floor without human assistance

8 Routine maintenance and servicing should be similar to non robotic equipment. Unit should be able to perform self diagnostics

9 Robot should run without any external navigational aids like bar codes, reflective tape or other devices that will modify a customer's building.

THE FIRST COMMERCIAL FLOOR CARE COMPANY THAT VENTURED INTO THE PRODUCTION OF ROBOTICS

Allen J. Bancroft
Robotics Project Engineer/Manager
The Kent Company
Elkhart, Indiana

Abstract

This paper will cover the history of innovation at the Kent Company, its acquisition by AB Electrolux of Sweden, Electrolux's vision that robotics would control the future, and the evolution of robotic floor care equipment. The paper will cover early prototypes, the learning curves associated with robotic floor care equipment development, bloopers, and the introduction of the RoboKent™ Scrubbervac to the cleaning industry.

Kent - An Historical Perspective

The central system vacuum patented by Gorden Kent eighty years ago launched the company which still bears his name. Kent, with his father and two brothers, began manufacturing in Rome, New York, in 1913. Innovations in floor care including portable vacuum cleaners and floor machines established the company as a viable force in the industry.

The Kent Company merged with Finnell Systems in 1962. Finnell was founded in Missouri in 1916 and relocated to Elkhart, Indiana, in 1928. Kent continued to manufacture in both New York and Elkhart until after AB Electrolux acquired the company in 1969. Operations were centralized in Elkhart in 1971 and Kent is now one of the oldest full line manufacturers of floor care equipment in the United States.

Product innovation has always been a Kent Company hallmark. Kent patented the offset design floor machine in 1929 and introduced the revolutionary MICROSTAT® vacuum which filters virtually 100% of bacteria-laden particles from the air for use in hospitals in 1959. With its acquisition of Kent, Electrolux reemphasized innovation in an industry still often associated with mops and buckets. "Floor Care Made Easy" is Kent's signature and, with a focus on robotics, the tag, "The Future Is Here" has been added.

Electrolux - An Historical Perspective

AB Electrolux, a Swedish company, is one of the world's leading producers of household appliances. It leads the European market and, as owner of Frigidaire, is the third largest producer in the US. The group is also the world's largest company in floor care products and absorption refrigerators for recreational vehicles and hotel/motel rooms. Outside of household appliances, the Electrolux group is one of the world's top two companies in the areas of food service equipment, industrial laundry equipment, forestry and garden equipment, refrigerator compressors, and car safety belts.

The Electrolux group has approximately 600 operating companies in more than fifty countries and, since the eighties, has been concentrating on investments in new product design to carry it into the future.

Robotic Development

Early in 1986, Electrolux became interested in the possibility of robotizing several different types of products. Electrolux had previously been involved with Unimation, a company founded to focus on the development of industrial robotics. Joseph Engelberger, considered to be the father of industrial robotics, founded Unimation in the late sixties and sold it to White Westinghouse in the early 1980s. When Electrolux contacted Engelberger in 1986, he had recently founded Transitions Research Corporation, a start-up company geared towards the development of mobile robotics. Engelberger's background also included the successful design of assembly robots for the country's major automotive manufacturers.

"Copyright © 1993 by Allen J. Bancroft. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission."

Electrolux asked TRC to look into the feasibility of robotizing lawn mowers, floor care equipment, and a household robot similar to "Rosie" on the TV cartoon, "The Jetsons." Electrolux envisioned robotic equipment of all types as the future in manufacturing and productivity. They knew it would happen, but they just didn't know how soon. What Electrolux did know was that it wanted to be the company known as the leader in robotic technological development and manufacturing.

It was decided to first concentrate on developing a robotic floor scrubber. Electrolux was so convinced that the future was in robotics that they sent two engineers from Sweden for a two year period to do technology transfer back to Stockholm.

The First Prototypes

When Electrolux decided that the floor scrubber would be the first robotic experiment, they directed The Kent Company to send TRC a standard 32" autoscrubber.

When the machine arrived at TRC, there were many meetings to determine where the drive motors would go, where the electronics would be mounted, how the machine would turn on a soapy floor, how to have the brush head and the squeegee go up and down, and how to design the machine for use in manual mode as well as in automatic mode. And those weren't the only major hurdles. The machine had to cope with people and other obstacles and, most importantly, still be able to do an excellent cleaning job. A robotic scrubber that moves around on the floor all by itself isn't any good if it doesn't clean as well as the traditional equipment.

Several months of engineering resulted in the birth of "Big Bertha." This machine marked the first attempt to robotize a piece of floor care equipment. Initially tested in a local school scrubbing floors, this first prototype did indeed prove Electrolux's vision that floor care equipment could be robotized. After more preliminary testing, "Big Bertha" was retired. Electrolux then decided that it was important to get marketing input on what a robotic scrubber should do. It was decided that the first autoscrubber, "Big Bertha," was too big to transport around the country for field tests. Therefore Kent sent TRC smaller autoscrubbers to be transformed into robots so that it would be easier to transport and test them in real life situations. TRC began robotizing three Kent 20" autoscrubbers early in 1988. These machines had 11-gallon solution and recovery tanks making them much lighter than the 32" machine with two 27-gallon tanks.

The first 20" autoscrubber was robotized and configured several different ways. Electronics were first installed on the side of the machine. It soon became apparent that that location was susceptible to damage on turns or when something bumped the side of the machine. Next the electronic components were mounted on the top of the machine. That wasn't feasible because it was too easy, while filling the tanks from the top, for overflow to run directly into the electronics box. Next we mounted the electronics on the front of the machine and found out that it was too difficult to protect them from bumps head-on. We finally decided that the most logical place for the majority of the electronic components was on the rear of the machine. Installing the electronics on the rear of the machine also made it very convenient for them to be serviced. The rear placement of the electronics is still the optimum choice.

We also went through several stages of deciding what the main obstacle sensors would be. An early prototype had a scanner mounted on the front of the machine. That didn't provide enough information and it was extremely susceptible to breakage.

Early on we mounted sensors on the right side of the brush head so that the machine could scrub right up to the wall. First we mounted a paddle that rubbed right against the wall, but end users didn't like the fact that the paddle sometimes marked the wall. We then went to a thick foam on the brush head that was sensitive and cleaned right up to the wall. That scuffed the wall too, so that idea was discarded. Our final wall-tracking device was a non-contact sonar sensor, similar to what is used in a consumer camera to measure distance.

At this point we had to face the fact that it would be best to design the cleaning equipment around the robotics (navigation)

rather than robotizing an existing piece of equipment. It's much better and easier to add the cleaning device once the vehicle navigates properly. Since TRC didn't specialize in cleaning, however, a completed vehicle had to be used to develop those early prototypes. Three prototypes were ready for alpha testing in early 1989.

Late Night Testing

At that point, Kent assigned a sales manager from the East Coast to monitor the progress of the robotic development and provide input on real life scenarios in the cleaning industry.

Those first tests launched a series of incidents that would fill a book of bloopers. We chose a local grocery store in Danbury, Connecticut, as the first testing site. Late night tests with this space-age robot caused quite a bit of excitement. We used a 24-hour grocery store so that we could do our tests in the early morning hours when fewer people were shopping. In one of the very first trials with the robot going down the aisle, a little grandmotherly-looking lady walked by the other end of the aisle with her grocery cart. Here we are at 1:00 a.m. in the morning with a piece of equipment that appears to be straight out of a science fiction movie. The little old lady pushing her cart across the end of the aisle notices the robot out of the corner of her eye, stops, turns towards the robot, and sees it coming straight at her. Then she looks up towards the other end of the aisle and sees fifteen people, gasping for air, hoping that she will move out of the way and let the robot pass. Instead she froze in place. I started sprinting down the aisle knowing we were going to have an emergency medical situation, that the robot would hit her. I just didn't know whether she'd have a heart attack first or be run over. Lo and behold, the robot planned its path right around her and continued cleaning the rest of the area without a mishap. We knew at that point we were definitely headed in the right direction.

A trip or two later to the same grocery store we were a little less lucky. As the robot was doing its last cleaning pass, it encountered a free-standing display of soup cans set up at the end of an aisle as a "special of the week." After our last experience, we didn't doubt for a minute that the robot would move around the display. That's when the corner of the brush head caught the bottom of the display and sent 400 soup cans rolling around the floor. Today we laugh. At the time, the store manager, Kent's salesman/consultant and I didn't think it was the least bit funny. Today we are very close to introducing robotic cleaning equipment for supermarkets based on some of those early tests.

Following those first several months of testing it was time to move on to the next step - limited production.

Making The Big Move

TRC was a tremendous help in developing the initial three prototypes, but Kent decided to develop a robotics department at its Elkhart location so that development and floor care equipment manufacturing would be at the same site. It seemed wiser to hire someone from TRC who was familiar with the project than to try to find someone new. New Years Day is always the time for new beginnings, so it seemed appropriate that on January 1, 1991, I moved to Elkhart to develop Kent's robotics program. TRC was pleased that someone who had worked on the project since its inception would continue because it made for excellent communication and interface from manufacturing to TRC.

Kent's Commitment To Robotics

The Kent Company shared the parent company's vision of robotics as the way to increase productivity in floor care. Although TRC would still work with us, it soon became apparent that we would have to develop a fully staffed department. The first hire was a software engineer. Since then our group has grown to a staff of seven full-time people dedicated solely to robotics development. Since Kent is an original equipment manufacturer of industrial cleaning equipment, it was necessary to hire the majority of the robotics staff from outside the company to acquire the talent we needed. The staff consists of technicians, software engineers, mechanical engineers, draftsmen, and electrical engineers.

Kent's Robotic Learning Curve

We learned invaluable lessons from the many mistakes and missteps that always result from a venture into the unknown. In the process of training a new software engineer, I told him to put the first prototype on a lift table. It was necessary to do some repair on the navigation system which had developed an intermittent connection problem. Although not completely familiar with the machine, he lowered the lift table, drove the robot up the ramp, and raised the lift table to its full five feet so we could do the work. While I took a phone call, he studied the electronics and patiently waited to get his hands on the machine for the first time . When I hung up the phone, he told me he thought he had discovered the problem. Not knowing that the machine was still on, I said, "great." What he didn't know was that disabling or unplugging the handle bar control at that time made the machine run full speed in reverse. As he leaned forward to show me the loose wire causing the problem, I panicked. Before I could warn him, he reached into the controls and touched the loose wire. It broke completely, and it ***was*** the problem all right. The machine went full speed in reverse, he on one side, me on the other, and the machine five feet up in the air. We tried to hold it but couldn't. The machine landed on its back full of water and batteries. We both sat staring at the only prototype machine we had, now a $50,000 pile of components, and wondered whether we would still have jobs the next day. With the machine in ruins, everyone in the group pitched in for a continuous eighteen-hour repair job. From that we learned to design the machine so that when the handlebar is unplugged, the machine will not move. Many of the built-in safety features actually came from real life experiences making today's RoboKent a real user-friendly robot.

Marriott's Involvement

With a robotics staff in place and development well on its way, it was time to think about selling robots. Electrolux had been talking with the Marriott Corporation about an exclusive agreement, a joint venture situation, in which Marriott would share in development costs as well as buy all of our prototypes and production for a period of two years. While Marriott is usually thought of in connection with lodging, that was only one of their smaller business segments. They are also involved in airport food services and managing cleaning facilities in hospitals, office buildings, and school districts. Once an agreement was reached in February, 1991, we started installing prototypes. We looked for local, visible sites. Marriott decided to start in Chicago where they had two big hospital accounts.

First Prototypes Installed At The End-User

The first machine was installed at Northwestern University in Chicago. As far as we know, this was the first commercial robotic floor care equipment actually sold and in place at an end-user location. All of the Marriott management team from the main office in Washington, DC, was there for what was to be history in the making.

This was to be another learning experience. As we unveiled the RoboKent scrubber for all of Marriott and the hospital personnel, you could feel the excitement permeate the air. Dozens of people watched as the robot went up and down the basement aisle scrubbing, cleaning, and drying in one pass. In fact, everyone stood quietly in awe as the RoboKent scrubbed the floor perfectly. On its final cleaning pass, a janitor, caught unaware, happened to walk by at the opposite end of the hall. He looked up in shock at the machine moving by itself and then he saw his boss standing with me on the far end of the corridor. He ran towards the machine shouting at the top of his lungs, "I'll get that run-away machine, Richard!". His boss yelled down the corridor, "Leroy, leave that robot alone," and Leroy responded, "What robot?" We laughed but this incident was repeated several times with other employees in other locations. That taught us to be very aware of the impact of robotic equipment in action on people and to be prepared for unusual reactions...something that we still caution people to consider and prepare for today.

After we installed Northwestern's machine and a machine at The Rush

Presbyterian Hospital in Chicago, we recognized some limitations.

The Chicago machines were hand built as prototypes, not necessarily machines to be left in the hands of an end-user. After a few trips to Chicago to repair loose electronic connections as well as to do repair on safety sensors, we knew further development was imperative.

Programming the machine was the next immediate challenge. When we first installed the machines, they were programmed by walking them down the hallways and entering data into their on-board computers as we went. That was great the first time we went to the hospitals. We programmed the several hallways that the customer indicated and left. Two weeks later the customer called back and reported that the robot was running fine; please come program the other thirteen floors of hallways. That task in itself took MANY days. We realized we couldn't go into production if we had to travel to every end-user location to program a robot for each and every hallway. Besides the fact that the initial installation could take several days, hallways always seemed to be under reconstruction or reconfiguration. That resulted in requests for us to come and reprogram those sections. There were too many limitations to list, but the reality was, manually teaching production robots for an industry where floor patterns can change from day to day was not going to work. It's not like an industrial robot where it's programmed once and the robot will perform the same task time and time again in an unchanging environment. In addition, installation costs could surpass the actual cost of the machine. This forced us to look into other programming methods and from that sprang the "Auto Learning" process, patent pending.

Patent Pending "Auto Learning"

The refinement of "Auto Learning" was the first major engineering victory for Kent's robotics department. I felt that if we could manually walk a machine to teach it and enter data into the robot's computer, the robot should be able to teach itself while it traveled the hallways, entering data into its own computer. I was told I was crazy by several well-respected people in the mobile robotics field. I saw then that this task was going to be all ours; no one else believed in it. Being told that a robot couldn't teach itself made me more determined than ever to make it happen. Two and a half months later we had the first prototype running in our warehouse in Elkhart.

That was just one of many major challenges that brought vigor and vitality to Kent's robotics department. It also brought Kent to the forefront of robotics technology. Now, knowing that we could install robotic scrubbers at an end-user's site without being required to go back to program the machines, we were ready to do a limited production run of prototypes, the third generation.

With the old-style programming, once hallways were programmed, the operator had to remember which program corresponded to what area. The operator needed to either remember hundreds of hallway numbers or he had to have some sort of chart mounted on the machine. Once we updated the software in the first two Chicago sites, the operators' gratitude knew no bounds. Unlike those who get a robot today, they really knew the old fashioned way.

After three months of testing with "Auto Learning" installed, Marriott decided they wanted ten more machines to spread around the country in various geographical locations and situations.

These ten machines were put into operation in late 1992 with the idea that we would get direct feedback from Marriott about how the machines worked in real-life environments, if they were user friendly, and, most importantly, their productivity quotients. The goal was to understand the expectations of end users and how, if at all, the next generation machines could be made more productive.

Prototype Testing Complete

After eighteen months of testing on the second and third generation machines, it was time to evaluate all input from the field in conjunction with the engineering ides that we had compiled.

We learned that a machine with an 11-gallon solution tank and an 11-gallon

recovery tank needed to be serviced approximately every thirty-five minutes. That certainly didn't increase productivity. We also found out that after four hours of run time the batteries needed to be recharged. That meant that the robot sat idle for up to nine hours during the recharge time. That didn't increase productivity either. The chemicals put in the solution tank couldn't be flushed out because there were no dump fittings on the solution tank. Again it was apparent that we needed to first develop the robotics and then add cleaning components. One big complaint was that the machines really looked like we took an existing autoscrubber and hung electronics on it. It was an unwieldy sight and not very eye appealing. Those ideas and views came from end users. Now it was time for the input from the robotic team's perspective.

We decided we needed a better design to make servicing the electronics easier. The original twelve machines, still in operation today, are serviced by our in-house technicians. Larger production runs would require making the electronic components more accessible and fail-safe for field service people to replace.

Generation Four

Following two months of information compilation, we were ready to start redesigning. It was finally time to engineer a machine from the ground up to become a totally robotic scrubber, not take a scrubber and make it into a robot.

We designed larger tanks, developed a slide-out exchangeable battery pack, and incorporated means for dumping both the solution and recovery tanks.

Production Set-Up

Building a robot in the laboratory was one thing. Building a robot in the manufacturing area was something else altogether. We selected three people out of our factory to be dedicated to the manufacturing of robotic equipment. We also had to set up a custom line conducive to the safety of handling electronics. We went into full line production in January of 1992. Although it took a few months, our production line now runs without the supervision of the robotics group. Factory personnel tell us that building the robot is just like building any other piece of equipment.

To date, we've assembled more than eighty robotic scrubbers on that assembly line. That line was also designed to support the manufacturing of our robotic power sweepers, robotic burnishers, and other robotic floor care equipment now in development and testing stages.

A secondary benefit of doing development at the manufacturing location is that there is plenty of room to test the robots after they've been assembled. Additionally, all robotic floor care equipment is tested for its quality of cleaning by factory personnel who are thoroughly knowledgeable about the cleaning industry's requirements and concerns.

Conclusion

Making floor care easier and more productive has long been The Kent Company's mission. To achieve that objective, we are devoted to staying at the forefront of technology by investing in long-range development. Productivity is accomplished by making the installation and use of robotic equipment easy for maintenance personnel who do not have engineering degrees at a cost that will generate a high return on investment in a short time span. With costs being squeezed from budgets and fewer personnel being asked to do more, The Kent Company believes that the development of robotic technology and equipment in the US will have dual benefits. More manufacturing jobs will be created and the amount of work maintenance personnel can accomplish within allotted time frames will be increased.

We believe that the future is robotics and that "The Future Is Here."

NON-GEOMETRIC HAZARD DETECTION FOR A MARS MICROROVER

Brian H. Wilcox
Supervisor, Robotic Vehicles Group
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Abstract

Non-geometric hazards (i.e., those which cannot be characterized soley by their shape, but instead are related to mechanical properties such as strength and friction) may pose a significant risk to planetary rovers. This paper describes a means for an articulated vehicle to detect sinkage and slippage in such material so as to prevent entrapment and to correct for dead-reckoning errors. Simulation results and preliminary indications of test data are described.

Introduction

For an exploring vehicle to move safely over the surface of another planet, it is potentially important to know if the vehicle is sinking into very soft surface material or is experiencing high levels of wheel slip. For example, at the Viking 1 landing site, about 14% of the surface is drift material, and one of the landing legs sank 17 cm into that material.[1] Previous studies of non-geometric hazard detection for planetary rovers, which assumed very large (~1000 Kg) rovers, have focussed on Ground Penetrating Radar to detect subsurface hazards.[2,3] However, mass and power constraints for microrover missions lead us to desire means to detect these hazards without requiring additional mass, power, or complexity beyond the basic vehicle configuration.

For the purposes of this discussion, we consider the mission model of NASA's Mars Environmental Survey (MESUR) Pathfinder project, scheduled for launch to Mars in November, 1996. In this mission, a microrover with a mass of under 10 Kg will traverse over the terrain within a few tens or hundreds of meters from it's lander to goal points selected by ground-based analysis of images taken by lander stereo cameras. These goal points are selected for their scientific interest, and it is important that they be approached quite accurately (for example to take a spectrum of a particular rock). Thus safety and improvement in the accuracy of dead-reckoning navigation are important reasons to develop a reliable means for estimating the sinkage and slippage of the rover wheels. Means for detection and avoiding geometric hazards are described elsewhere.[4]

The Pathfinder rover is a six-wheeled "rocker-bogie" articulated vehicle. It will be functionally equivalent and the same size as our research vehicle "Rocky 3.2", shown in Figure 1.

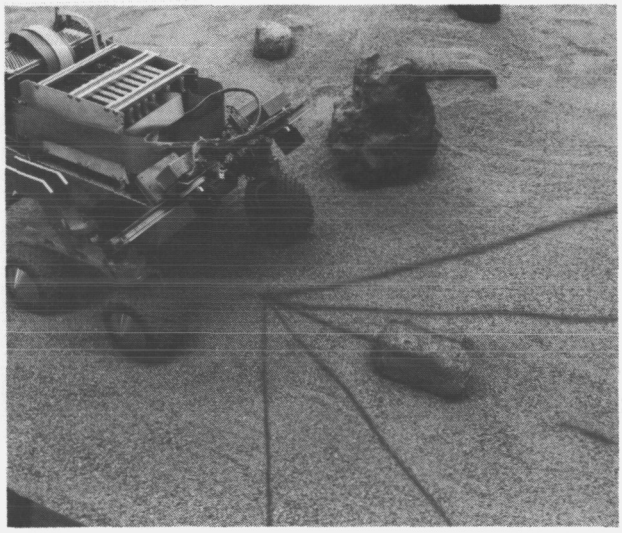


Figure 1. Rocky 3.2 vehicle with laser stripe sensors

Rocky 3.2 (one of a long line of Rockys) has sensors for wheel speed (all wheels are driven) and for determining the articulation angles of the chassis. (The articulations are passive so that each wheel follows the terrain contours independently.) It also has a look-ahead ranging sensor based on detecting, in a CCD image, the position of laser stripes projected ahead of the vehicle (Figure 1 is reproduced from a color original with a blue filter so the red laser stripes show as dark lines). Because the computation on-board the rover is very limited (an 8085 CPU, about 20 times less powerful than a typical personal computer), it is important

-Copyright c 1993 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved

that only a small amount of sensor data be taken and processed. Thus it is important to formulate simple algorithms for estimation of slippage and sinkage, and to do performance evaluation based on the concept that only the wheel, chassis, and a minimum number of discrete measurements from the look-ahead sensor can be used as input to the system. If a simple algorithm gives good performance, in terms of improvement of dead reckoning vs basic odometry and in detection of hazardous sinkage conditions, then the increased computational load will be justified.

The Sinkage and Slippage Model

We consider a planar model as shown in Figure 2. Specifically, there are three wheels connected with passive but instrumented linkages so that they remain in contact with the soil as they roll. By processing the pitch and articulation sensor values we can compute the difference in elevation between the rear wheel and the center or front wheels (call these z(1) and z(2), respectively). We also have a look-ahead ranging sensor which examines a number of discrete points on the ground ahead of the vehicle. Again, by processing the sensor data, we can compute the elevation difference between the rear-wheel nominal contact point and the elevation of each sensed point on the ground ahead of the vehicle (call these z(3) ... z(N)). Needless to say, all these measurements have noise which must be accounted for in the analysis.

Assumptions

We assume that undisturbed terrain in this planar model has an elevation function y(x), where y is the elevation at a point x along the horizontal axis. When the vehicle moves ahead, the front wheel sinks in the soil by an amount s(x), so that it rolls along in contact with the function y(x)-s(x). We assume that the trailing wheels do not further compress the soil (since the wheel loading of this vehicle is roughly uniform). Thus they also track y(x)-s(x). This is a key assumption which, if not approximately correct, will lead to a general failure of the entire approach. If the wheels all turn at the same rate (which is reasonable since they are geared so low that in normal terrain they run effectively at the no-load speed), then when the wheel circumference has moved a distance w the vehicle will advance some distance x in the horizontal direction, usually less than w, due to wheel slippage. This slippage will generally be a function of the type and slope of the soil.

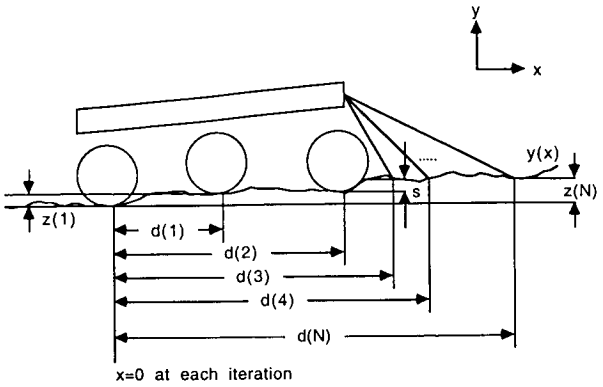


Figure 2. Planar model and symbol definitions

Objective

The objective of this analysis is to estimate x and s(x) given the "odometer" reading w, the values of z(1)...z(N), and the associated measurement noise v(1)...v(N). Intuitively this should be possible, since if y(x) and s(x) were known exactly up to the forward-most sensor (a ranging sensor for y and the front wheel for s), then for a given Δw, there would usually be a unique Δx which would allow all the sensor readings to match their predicted values. In other words one would "slide" the rear and center wheels along the curve y(x)-s(x) until the observed elevation difference z(1) is matched between $x+\Delta x$ (the new position of the rear wheel) and $x+\Delta x+d(1)$ (the new position of the center wheel), which would fix Δx. Then one would use the measured elevation of the front wheel to compute y-s at that point (thereby extending our knowledge of s(x) forward by Δx. Similarly, we would use the measured elevation at the forwardmost range sensor to extend our knowledge of y(x) by Δx. This process would repeat so as to build an arbitrary sequence of Δx, s(x+d(2)), and y(x+d(N)) values. We would, of course, assume a $y_0(x_0)$ value as the starting elevation and position of the rear wheel. (Knowledge of the initial y(x) and s(x) functions between x and x + d(N) is trivial since the vehicle will disembark from the lander along a ramp of known geometry and with negligible slip and sinkage.)

One potential problem with this approach is that values of z(1)...z(N) will not be taken densely along the vehicle trajectory. Actually, the processor on the vehicle is sufficiently slow and burdened with other activities that the navigation and mobility sensors are only monitored roughly every wheel radius of forward traverse. This is often enough to ensure that rocks, craters and other geometric hazards can be detected and avoided (an issue not addressed in this paper).

There are several issues which need to be considered with this model. First, if $\mathbf{Z}(k)=\text{col}(z(N),...,z(1))$ at cycle k is measured on terrain which is very flat (compared to the measurement uncertainties $\mathbf{V}(k)$) then we would still like to have a reasonable estimate of forward travel. This suggests that we should have a prior model of the distribution of the slip x(w), and that we should form a Maximum A Posteriori (MAP) estimate of the slip.[5]

Following our heuristic argument above, if we were to "slide" the vehicle along until the observed elevation difference z(i) is matched, this corresponds to generating a discrete set of values y(i), i=0,...M1 and s(i), i=1,...M2 which can be thought of as our best estimate "histograms" (i.e., discretized piecewise constant representations) of the y(x) and s(x) functions. The horizontal density of these estimates should be sufficiently great to allow accurate models of the terrain for purposes of simulation, but not so great as to unduly burden the processor. Since the wheels mechanically average the terrain over a length equal to the tire contact patch (about a third of a wheel radius) we would tend to discretize the model at about this level. Thus we might have M2=30 or so and M1=60 or so (the actual Rocky 3.2 vehicle has 13 cm dia. wheels and an overall length of 60 cm, with the look-ahead sensor reaching about one vehicle length).

Thus we can now outline a procedure for estimating the sinkage and slippage of the microrover:

1) Measure the elevation differences z(1)...z(N).

2) Use previously-estimated (described below) histograms y(i), i=0,...,M1 and s(i), i=0,...,M2, as well as a Gaussian prior distribution for Δx with mean m_x and variance σ_x^2 to compute the (nonlinear) MAP estimate for Δx. We assume the distribution for measurement noise for each z(i) is also independent and Gaussian. Since the MAP estimate of independent Gaussians is a weighted least-squares estimate, we compute:

$$\min_i\Big(\sum_{j=3}^{N-1}\big[(1/\sigma_{z(j)}^2)(z(j) - y(d(j)+i) - y(i)+s(i))^2\big] + (1/\sigma_{z(1)}^2)(z(1) - y(d(1)+i) + s(d(1)+i) - y(i) + s(i))^2 + (1/\sigma_x^2)(i-m_x)^2\Big)$$

The interpretation of this expression is as follows: to maximize the posterior probability, which is the product of exponentials, we need to minimize the magnitude of the exponent. If we let i be the histogram bin which we assume the rear wheel has advanced to (and changed to an elevation y(i)-s(i)), then the summation from j=3 to N-1 is of squared errors between the ranging sensor elevations and the corresponding y values in the histogram. The next term is the weighted squared error for the middle wheel, incorporating the histogram data for s(1) as well as y(1). The last term is from the Bayesian prior distribution. Note that z(2) does not even appear in this expression, as the advance of the front wheel involves an unknown amount of sinkage in the soil and so there is no histogram data with which to compare. A similar situation arises with z(N) in the summation, since y(x) is unknown ahead of the forwardmost sensed point.

We implicitly assume that the forward advance is not so great as to push the next sensed point z(N-1) off the end of the histogram, although this could be accounted for if necessary. We would then perform a parabolic interpolation of the weighted-sum-of-squares to get a refined estimate of Δx to a fraction of a histogram bin. While not strictly valid, interpolation of the error function should be better than taking integer bins, while not as computationally intensive as the more conceptually-correct approach of computing the minimal error function on interpolated data. Note also that we could compute m_x as a function of the data here prior to finding the minimum over i to account for the fact that our expected slip is a function of average terrain slope. For example, we could compute

$$\Big(\sum_{j=1}^{N}(z(j)/d(j))\Big)/N$$

as an estimate of the slope and compute some linear or non-linear function of this to compute m_x. We could also modify the estimates of m_x and σ_x^2 using prior estimates to adapt and refine our Bayesian prior.

3) Now that we have an estimate for Δx, we translate the histograms for y and s forward by Δx and up by $y(\Delta x)$. This requires interpolation, due to the non-integer nature of Δx, so we assume that linear interpolation between adjacent points is adequate (again to reduce computational complexity). We extend our knowledge of y forward by linear interpolation from the translated old y(N) value to the observed z(N) at d(N). Similarly, we extend our knowledge of s forward using linear interpolation from the translated s(d(2)) to a new forwardmost value s(d(2))=y(d(2))-z(2).

4) We need some way to incorporate the new measurements into the histogram for y (otherwise only the forwardmost measurement y(N) will play a role in defining the function, which seems to waste a great deal of valuable information). Note that between the old d(N) and the new d(N) we have a linear approximation to y(x). When the vehicle moves forward by Δx (generally less than d(N)-d(N-1)), we will get a new value for y from z(N-1) which will, in general, not lie on the previous linear approximation to y. Since we expect that our measurement noise $\sigma_{z(N-1)}$ will be quite small compared to the grossness of the linear interpolation, we would like to force the histogram to conform to the data at this point (the new d(N-1) point). We would also expect y(x) to be a continuous function, so that nearby points should also be modified. For simplicity, we will *assume* that adjacent histogram bins will be updated by "splitting the difference", i.e. they will be reassigned values halfway between the new measurements of y based on each of the z(i) measurements for i<N and the old (but translated) histogram value. This is an ad-hoc assumption made in the interests of computational simplicity which will hopefully allow a fairly accurate estimate of y(x) to be generated as all of the sensors sweep over the surface. We can perform a corresponding process for s(x) by assuming that deviations between z(1) and y(d(1))-s(d(1)) are due to errors in the measurement of s and not y, which makes some sense because by this time the histogram for y has been refined with multiple measurements while the histogram for s has been generated only by piecewise linear interpolation out to the single measurement at z(2) (i.e. the front wheel).

5) Lastly, move the vehicle forward and repeat the cycle.

This model and analysis are very simple and somewhat suspect from a theoretical point-of-view. However, as in many practical applications, real-time performance and computational complexity are of paramount importance, with the alternative being not to do any estimation at all. Thus we would like to know what the performance of this simple estimation procedure is, and to what degree it gives improvement over use of the prior mean m_x to estimate over-the-ground distance travelled and not estimating sinkage at all (and accepting the risk of getting stuck). We would also like the evaluate the usefulness of having more ranging sensor measurements as opposed to fewer, since each additional measurement has cost and may only be needed for this purpose (as rocks and craters may be detectable with as few as two look-ahead range points). If possible, we would like to also have a way of choosing the distances d(3)...d(N).

Thus what remains to be done is 1) perform an evaluation of the performance of the system by estimating the variance in the slippage and sinkage estimates by Monte Carlo numerical simulation (since the nonlinear MAP formulation is intrinsically iterative and because we want to explicitly incorporate the effects of quantization into the histogram bins, the effects of resampling and interpolation, etc.). This simulation will evaluate the effects when the data are not drawn from a Gaussian distribution, such as a uniform distribution of equal or different mean. Lastly, we would like to evaluate the effect on performance of varying the number of sensed values N, of modifying the mean m_x of the prior slip distribution based on experience, and of changes in the sensor noise $\sigma_{z(i)}$, which we might adjust in an ad-hoc way to account for the aliasing which the point-range measurements will have in estimating the average elevation over the histogram bins, where the spectrum of y(x) might grossly violate the Nyquist sampling theorem when binned in this manner.

The assumed model for y(x) in the simulation needs to be chosen with some care. A scale-invariant (fractal) model is attractive, but we need to recognize that the hazard-detection and avoidance system will effectively clip the distribution of terrain features at some particular scale. Similarly, a model for s(x) needs to be formulated, which will be slowly varying and of low amplitude. It would be good to assess the performance of the system when the slippage and sinkage are correlated, as one would intuitively expect, even though the model does not incorporate that effect (although it easily could). Another interesting correlation which would be good to model in the simulation is the fact that the mechanical linkages in the vehicle chassis cause the noise in the measurements z(i) to be highly correlated (since wheel pairs are at opposite ends of links), even though they may be Gaussian (from digitizing analog potentiometer values or peak detection in analog CCD scan lines).

The simulation model for y(x) and s(x)

As mentioned above, we desire to test the sinkage and slippage estimation algorithm on terrain which is "scale invariant". Specifically, we wish to create a sample random terrain in the form of a histogram (i.e. sequence) at the same resolution as that maintained by the estimation algorithm. This is accomplished by uniformly sampling a linear combination of sine waves, whose amplitude is random over a uniform range extending from zero to some fixed multiple of the wavelength (thereby ensuring scale invariance), and whose phase is random over $[0,2\pi]$. Twenty different wavelengths are combined over the range from 1 cm to 1.9 meters, with each one 30% longer than the previous one. This range encompasses all scales of interest: smaller scales average to zero over the bins and longer scales are virtually flat over the length of the vehicle and its look-ahead ranging sensor. (Note that the smaller scales will exhibit substantial aliasing when binned, which is an important and real effect that needs to be modelled by the analysis.) As mentioned before, a "smooth" simulated terrain is realistic here, since the geometric hazard detection system will avoid rough or discontinuous terrain.

The terrain we construct here is characterized by a single parameter: the maximum slope of each sine wave component. We call this parameter the "roughness" of the terrain. Both y(x) and s(x) are created by this technique, but s(x) is clipped at zero so that only positive values of sinkage are allowed. The "estimated" histograms of y and s are initialized with the "actual" values from this simulation; from that point on the estimation procedure extends them. This is reasonable since, as mentioned above, the first meter or so of traverse will be on the lander exit ramp and therefore known. We arbitrarily set the roughness of the sinkage function s(x) to be 20% that of y(x), based on the philosophy that the terrain mechanical characteristics are more slowly-varying than the surface topography.

It is perhaps worth mentioning that the approach of combining sine waves over a large number of different scales is computationally intensive, but need only be done once to simulate a large number of different terrain types, since to change the "roughness" only requires rescaling the vertical coordinate of a "standard" terrain, i.e. with unity roughness. Another approach to generating scale-invariant terrain, the use of Gauss-Markov random sequences, needs to be fairly high-order to get the needed range of scales and thus becomes extremely complex to analyze.

Specifics parameters for initializing the model are drawn from the actual design of the Rocky 3.2 microrover. Thus, for example, the distances from the sensed points to the rear wheel contact point are 25, 50, 60, and 80 cm for the middle wheel, front wheel, downlooking range sensor, and outlooking range sensor, respectively. The sensor noise (standard deviations) associated with these elevation differences are 0.04 mm for the wheel sensors, and 2 mm for the look-ahead sensors. We normally expect the vehicle to advance about 5 cm in each sensing cycle.

Simulation Trials

For each trial run, we evaluate the odometry error and sinkage error as a function of bin size and terrain roughness for different input assumptions. We evaluate bin sizes from 0.2 cm to 8 cm, which spans the range from very fine to very coarse compared the the expected forward advance per cycle. We evaluate terrain roughness ranging from a maximum slope at each scale of 0.25% to 8%, which spans terrain from

very smooth (with typical elevation differences of 2 mm over the length of the vehicle) to very rough (with 8 cm typical elevation differences over the length of the vehicle, about the limit which the hazard avoidance system would permit). The simulation covers 62.5 meters of simulated terrain (25,000 bins at the finest bin size), which is created once and then resampled for the different simulations so that the effects of aliasing can be evaluated on identical terrain.

One important issue not addressed in the previous description of the algorithm is the choice of the search range for the weighted-sum-of- squares (WSS). Initially, the search was extended to $4\sigma + 4$ bins beyond the Bayesian prior mean. However, it was found that the simulation would occasionally get "stuck" and fail to advance the rover by the proper amount for several cycles, whereupon the simulation lost track of the terrain (i.e. presumably the internal histogram for y(x) had no relation to the actual y(x)). This was caused by the global minima of the WSS function not corresponding to the actual forward advance. A simple fix for this problem was to compute the secondary minima, and if it was beyond the global minima and nearly as good (within a factor of 3), then the search range on the next cycle was extended to include that minima. Note that, in all cases, the global minima is chosen for the simulation, and only that the search range is extended if another minima shows promise, so that it can be selected as the global minima on the next cycle. This effectively cured the problem, and subsequently the simulation was not observed to lose track of terrain.

Since we expect the look-ahead ranging sensor to have much worse measurement accuracy than the chassis articulation sensors, we will characterize the slippage estimation with the articulation-based elevation sensing noise, while the sinkage is based on the look-ahead sensor noise.

Thus we represent the results of this analysis by plotting the sinkage or slippage error against the terrain roughness value. Typically we would expect to have little or no cumulative error when the terrain is very rough, and if the terrain is smooth the algorithm will just return the Bayesian prior mean value as the result, so the error that accumulates is just the difference between the Bayesian prior mean and the actual mean value. Thus for slippage, for example, if the Bayesian prior is in error by 20% (that is, the actual expected distance advanced per sensing cycle is different from the Bayesian prior mean by 20%), then we would expect the algorithm to smoothly transition from small error to 20% error in estimating traverse distance as the roughness is increased from zero to a large value. We wish to establish the nature of this curve for both slippage and sinkage. Furthermore, to reduce computational complexity, we wish to determine how coarse the histogram bin size can be without excessive degradation of these results.

Table 1 shows the program output for the first test case, where the Bayesian prior overestimates the forward advance by 20%. Each entry in the table is the percentage odometry error over the 62.5 meter course, followed by the RMS sinkage error in parenthesis (in cm). Note that, indeed, the odometry error more-or-less smoothly falls from 20% for smooth terrain to near zero for rougher terrain. Furthermore, note that the performance improves as the bins get larger up to a point, and then declines for larger bin sizes, especially on rough terrain.

The two effects which seem to be occurring are severe aliasing for large bins (when the bins are larger than the advance of the vehicle), and poor terrain modelling for small bins. The former effect is compounded by the fact that we cannot fit a parabola to the WSS function if the minima is at zero bins of advance, since we do not compute the function for negative advance and so cannot bound the integral minima with values on each side, as needed for a parabolic interpolation. In this case we merely set the forward advance estimate to exactly zero. For large bins (e.g. 8 cm when the expected forward advance is 5 cm) this occurs commonly, and is only sometimes compensated for in later cycles. This produces a strong tendency to underestimate the distance travelled.

For very small bins, on the other hand, the algorithm we have selected for modelling the sparsely-sampled terrain is inadequate. Remember that we incorporate new z[i] measurements into the histogram by forcing the value at bin i to be consistent, and then "split the difference" on the i-1 and i+1 bins. When the bins are very fine this will produce narrow

SLIPPAGE ERROR AS A FUNCTION OF ROUGHNESS AND BIN SIZE
(each entry percent odometry error, RMS sinkage error (cm))

bin	roughness -- max slope at each scale											
(cm)	0.0025		0.0050		0.0100		0.0200		0.0400		0.0800	
0.25	20.87	(0.2)	15.24	(0.3)	10.36	(0.6)	6.62	(0.9)	3.57	(1.5)	-0.83	(6.3)
0.50	20.58	(0.2)	15.73	(0.3)	9.02	(0.5)	5.16	(0.8)	2.58	(1.4)	-1.20	(4.5)
0.75	18.94	(0.2)	13.57	(0.3)	7.86	(0.7)	3.69	(0.8)	1.84	(1.4)	-0.46	(3.4)
1.00	20.98	(0.2)	14.37	(0.3)	8.45	(0.4)	3.94	(0.7)	1.99	(1.2)	2.27	(2.8)
1.25	21.56	(0.2)	15.63	(0.3)	7.33	(0.4)	2.91	(0.6)	2.05	(1.3)	-2.17	(4.6)
1.50	19.30	(0.2)	11.15	(0.3)	5.56	(0.4)	1.42	(0.6)	0.10	(1.0)	-0.63	(2.9)
1.75	18.16	(0.2)	9.80	(0.3)	3.78	(0.4)	0.35	(0.5)	-14.61	(6.2)	-1.07	(2.6)
2.00	18.74	(0.2)	11.66	(0.2)	5.51	(0.4)	1.39	(0.4)	-0.32	(1.0)	-1.04	(2.9)
2.25	16.30	(0.2)	11.07	(0.2)	5.10	(0.3)	1.32	(0.5)	-0.01	(0.9)	-0.04	(2.2)
2.50	17.52	(0.2)	11.44	(0.2)	6.75	(0.4)	3.79	(0.6)	1.67	(1.2)	1.67	(2.9)
2.75	16.29	(0.2)	11.02	(0.2)	6.09	(0.3)	2.29	(0.6)	-2.21	(2.0)	2.01	(3.4)
3.00	12.97	(0.2)	7.60	(0.3)	1.97	(0.4)	-2.03	(0.6)	-4.16	(1.1)	-3.48	(3.0)
3.25	15.20	(0.2)	8.33	(0.2)	1.01	(0.3)	-3.58	(0.6)	-7.01	(1.4)	-4.99	(3.5)
3.50	15.98	(0.1)	10.44	(0.2)	5.64	(0.3)	1.14	(0.5)	-0.88	(1.2)	0.63	(3.3)
3.75	14.47	(0.2)	7.45	(0.2)	0.41	(0.3)	-5.57	(0.6)	-8.25	(1.5)	-4.91	(3.6)
4.00	14.82	(0.1)	8.97	(0.2)	2.74	(0.3)	-2.24	(0.6)	-4.85	(1.3)	-3.43	(3.5)
4.25	12.83	(0.2)	4.81	(0.2)	-5.49	(0.4)	-12.58	(0.8)	-14.81	(1.9)	-14.12	(4.0)
4.50	15.14	(0.2)	7.66	(0.2)	-0.33	(0.3)	-8.00	(0.7)	-8.91	(1.7)	-7.28	(4.0)
4.75	14.61	(0.2)	8.82	(0.2)	1.41	(0.3)	-5.23	(0.6)	-9.01	(1.4)	-8.88	(3.2)
5.00	15.04	(0.2)	9.75	(0.3)	4.66	(0.4)	-0.94	(0.8)	-2.37	(1.8)	1.74	(4.8)
5.25	12.31	(0.2)	3.32	(0.2)	-7.19	(0.4)	-15.85	(0.8)	-18.43	(1.8)	-13.87	(4.6)
5.50	12.66	(0.2)	6.01	(0.3)	-4.00	(0.4)	-12.23	(0.8)	-15.58	(1.6)	-14.52	(3.5)
5.75	14.54	(0.2)	6.20	(0.2)	-3.20	(0.3)	-13.75	(0.7)	-15.86	(1.6)	-13.18	(3.5)
6.00	16.26	(0.2)	9.72	(0.2)	1.25	(0.3)	-6.40	(0.8)	-8.48	(1.7)	-4.22	(4.1)
6.25	14.44	(0.2)	10.00	(0.3)	1.46	(0.4)	-4.67	(0.9)	-5.12	(1.8)	-0.16	(4.7)
6.50	10.56	(0.2)	-2.75	(0.2)	-18.84	(0.5)	-27.70	(1.1)	-29.74	(2.4)	-51.35	(5.8)
6.75	11.23	(0.2)	-2.80	(0.3)	-19.46	(0.4)	-29.11	(0.8)	-28.83	(1.7)	-54.61	(3.5)
7.00	11.60	(0.2)	0.36	(0.3)	-16.44	(0.4)	-25.05	(0.8)	-26.24	(1.6)	-23.42	(3.8)
7.25	14.54	(0.2)	1.71	(0.2)	-15.08	(0.5)	-25.49	(1.1)	-28.44	(2.6)	-27.28	(5.8)
7.50	15.21	(0.2)	3.99	(0.2)	-12.62	(0.5)	-23.38	(1.2)	-25.50	(2.7)	-22.45	(5.3)
7.75	13.48	(0.2)	4.52	(0.3)	-13.74	(0.4)	-24.38	(0.7)	-25.00	(1.8)	-30.19	(4.1)
8.00	17.31	(0.2)	6.79	(0.2)	-12.49	(0.4)	-21.23	(0.8)	-17.86	(1.8)	-15.76	(4.7)

Simulation Parameters
Actual mean advance per cycle: 5.0 cm, Sigma: 1.00 cm
Bayesian prior mean advance per cycle: 6.0 cm, Sigma: 0.05 cm
Unit-Roughness Terrain RMS Amplitude 1.11 meters over 62.5 meters

Statistical Attributes of Unit-Roughness Simulated Terrain by Bin Size
(each entry RMS bin-to-bin elevation change (cm),
RMS error in bin-to-bin linear projection (cm))

Bin Size (cm) X	X.00		X.25		X.50		X.75	
0.--	0.00	0.00	2.40	1.29	4.62	4.07	6.58	6.38
1.--	8.30	7.47	9.90	7.85	11.50	8.12	13.14	8.74
2.--	14.81	9.40	16.50	10.14	18.18	10.82	19.86	11.50
3.--	21.52	12.51	23.16	13.81	24.77	14.78	26.40	15.74
4.--	27.87	16.84	29.73	18.87	31.38	20.58	33.06	22.26
5.--	34.70	24.27	36.33	26.42	38.02	28.56	39.60	30.45
6.--	41.24	32.49	42.69	33.57	44.22	35.31	45.69	37.22
7.--	47.24	39.24	48.90	41.42	50.43	43.60	51.91	45.52

Table 1

"spikes" in the histograms, and not at all correspond to realistic terrain. The proper fix for this would be to "remember" when and where each prior measurement was taken, and try to perform a statistically-valid terrain reconstruction (based on some assumed terrain Fourier spectrum), incorporating all prior measurements and their uncertainties. However, this would be computationally demanding, and the procedure we have adopted seems to work quite well for intermediate-sized bins, about 2 cm long.

Note that the sinkage estimates in Table 1 are all about the same for a given roughness, and increase more-or-less proportionally to roughness. This is intuitively pleasing, since the high accuracy of the wheel sensors compared

to the look-ahead sensors makes the estimate of the forward advance of the vehicle (i.e. the minima of the WSS function) almost entirely a function of the wheel sensors. Thus, the primary function being estimated accurately is the loadbearing surface y(x)-s(x), with both y and s being much more uncertain than their difference. Then sinkage is estimated using the look-ahead sensor(s), with their large attendent noise. This suggests that a more appropriate implimentation for the actual vehicle is to use the wheel sensors alone to estimate travel along the loadbearing surface, and to use only one look-ahead sensed value to estimate sinkage. Thus it is irrelevant to examine the case of additional look-ahead sensing values so long as their noise is very large compared to the chassis articulation sensing. The "roughness" scale used corresponds approximately to the RMS elevation differences in meters over the scale of the vehicle, i.e. a roughness of 0.08 gives 8 cm of typical elevation difference across the vehicle. At the bottom of Table 1 is a chart showing some of the statistical properties of the unit-roughness simulated terrain: the RMS bin-to-bin elevation change and the RMS error in a bin-to-bin linear projection to the next bin, each as a function of bin size. This table has cm of bin size along the left, with fractions of a cm along the top. Note that the values for zero bin size, which in fact don't exist, are set to zero for printing purposes.

There is one striking fact represented in Table 1: we have selected the standard deviation of the Bayesian prior to be 0.05 cm (1% of the actual advance), when the sigma of the actual vehicle advance per cycle is 1 cm. This artificially "overweights" the Bayesian prior to show the smooth transition from 20% error to small error as the terrain gets rougher. However, the chassis articulation sensors are so accurate (σ=0.04 mm) that we can do much better than this. Figure 3 shows the results for different values of the Bayesian prior (1% and 10% of the actual). As one can see, with lower confidence in the prior, even on smooth terrain, the results are very good for bin sizes of about 2 cm (ranging from 5% error on very smooth or rough terrain to under 1% error on moderate terrain). This, again, is to be expected, since even the smooth terrain has large excursions compared to the sensor noise. If we reduce the prior variance further, however, the estimator performance degrades rapidly. This presumably occurs because much more error exists in the terrain histogram reconstruction than would be apparent from the sensor noise alone. Thus, if the simulation is not "driven" strongly by the Bayesian prior, it is "free" to choose any match to the sensor data, weighted artificially heavily due to the low sensor noise. Thus, even though the sensors are good, the terrain estimates which result from our sparse sampling and crude interpolation are not nearly so good. Thus weighting the perceived errors from this function by one over the sensor variance is unrealistic; we compensate by making the Bayesian prior very tight. Thus there is no particular value to be gained in evaluating somewhat different levels of sensor noise.

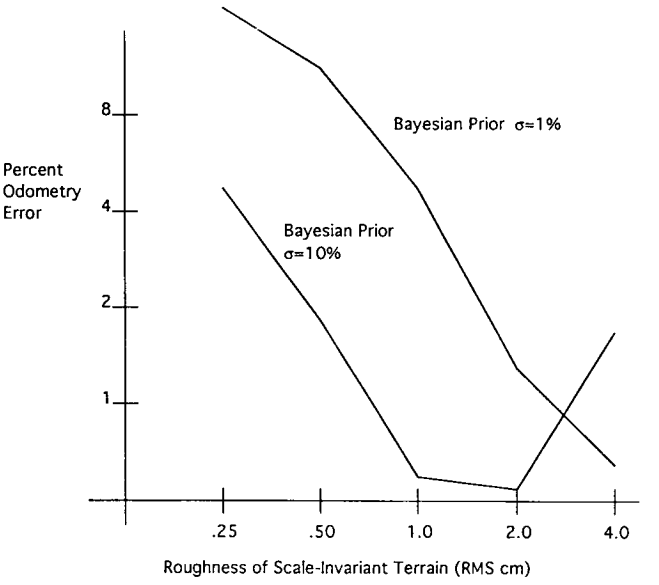


Figure 3. Odometry error as a function of terrain roughness (20% actual slip)

Numerous additional runs analogous to Table 1 were performed using different simulated terrain (using different seeds for the random number generator), and the results were virtually identical. Note that there are occasional anomalies where the performance is poor (such as in Table 1 at roughness 0.04 and bin size 1.75 cm). These anomalies presumably result when the terrain and binning processes conspire to give ambiguous terrain for matching purposes. This is to be expected, but so long as it is rare and does not give worse estimates than doing nothing (i.e. using just the prior mean estimate), then no harm is done. This is another reason to overweight the prior distribution.

Additional runs explored several issues. For example, when the prior distribution underestimates the forward advance, the performance is generally good for bin sizes between 2 and 3 cm, but that very bad performance is not uncommon. Another issue considered was the estimation performance when the actual forward advance is not Gaussian. Once again, the performance was excellent. Lastly, we considered the system performance when the actual slip is a function of sinkage and slope, as one would expect. The results were evaluated for the case when the mean of the actual advance per cycle drops linearly with increasing sinkage and/or slope, (and continuing with the non-Gaussian uniform actual distribution). Since the very rough terrain will probably have slopes and sinkages which would literally stop the vehicle under such an assumption, we clipped the left end of the uniform distribution at zero advance per cycle, so that the simulation doesn't get in an infinite loop (as would the actual vehicle). Here we have assumed that the linear coefficients are such that a 60% grade will stop the vehicle, as would sinkage of 5 cm. The performance on smooth terrain was poor, as the Bayesian prior of 6 cm/cycle was much larger than the actual average, which is 5 at best and 1 at worst, depending on terrain conditions. However, as soon as the roughness increases to 1 cm or so over the length of the vehicle, the odometry performance improves to within 10% and at 2-4 cm roughness. Only a few percent of odometry error is observed for bin sizes between 2 and 3 cm. This performance is very encouraging considering the simplicity of the model and the gross deviations which "reality" makes with the assumptions underlying the model.

Computational Requirements

As described above, the optimal bin size is in the neigborhood of 2.5 cm, which means that there are only 20 bins of data over the length of the vehicle to be accumulated and maintained, so the compuation and storage requirements are small. Good performance can be anticipated with only 2 terms in the WSS function-- one for the Bayesian prior (which can be precomputed and stored in a table) and one for the center wheel, since the look-ahead sensor is so noisy as not to affect the forward-advance estimate. (The front wheel moves onto unknown terrain, and so is not used in the matching process.) Since squaring can also be accomplished as a table look-up, the computation is of the order of 1 add and 2 table look-ups per bin, with typically 5 bins searched. Finding the global and secondary minima requires roughly 2 comparisons per bin. Maintenance of the histogram requires a relatively few operations also, since the histogram data can be in a ring buffer with a pointer, to avoid actually shifting the data in an array. Thus only the linear interpolation and "split the difference" operations are needed, which are simple. This implies that, with of the order of 100 operations per cycle, the odometry estimates of the vehicle can be markedly improved, and sinkage estimates provided.

Preliminary Test Results

The algorithm described above has been implemented on Rocky 3.2 and, as of this writing, a few test runs have been conducted. The preliminary indication is that the articulation sensor noise is substantially larger than anticipated, leading to odometry results which are somewhat degraded compared to the simulations. However, it appears that, even with the noisy data, the algorithm will give a very reasonable hazard alarm for sinkage and slippage. (In this case, we set the confidence in the Bayesian prior to be very high, and then threshold the WSS function to trigger a slip alarm.) Work is continuing on reducing the noise in the analog-to-digital portion of the system. Extensive tests for this system are planned for 1994.

Conclusions

The MAP estimation procedure developed here, based on a simple weighted-sum-of-squares computation, seems to give sinkage and slippage estimates which will allow planetary microrovers to detect and avoid a wide range of non-geometric hazards. Simulations suggest that it may also improve odometry markedly over simple wheel revolution counting, and thereby lead to a significant improvement in dead-reckoning accuracy for this class of vehicle.

Acknowledgements

The research described in this publication was carried out by the Jet Propulsion Laboratory,

California Institute of Technology, under contract with the National Aeronautics and Space Administration. The assistance of Jack Morrison and Tam Nguyen in implementing and testing the algorithm on the research vehicle is gratefully acknowledged.

References

[1] Moore, H. J. et al, "Surface Materials of the Viking Landing Sites", *Journal Geophysical Research*, v. 81, pp 4497-4523,1977.

[2] Douglass, R. J., "Surface Property Determination for Planetary Rovers", JPL Contract 958706 Quarterly Report, Martin Marietta Corporation, Denver, CO, 16 April, 1990.

[3] Olheft, G. R. and Strangway, D. W., "Electrical Properties of the Surface Layers of Mars," *Geophysical Research Letters*, Vol. 1, #3, July 1974.

[4] Wilcox, B. H., "Robotic Vehicles for Planetary Exploration", *Journal of Applied Intelligence* 2, p181-193, 1992.

[5] See, for example, Mendel, J. M. Lessons in Digital Estimation Theory, Prentice-Hall, Englewood Cliffs, NJ, p114.

AIAA-94-1277-CP

LOW COMPUTATION VISION-BASED NAVIGATION FOR A MARTIAN ROVER

Andrew S. Gavin

Massachusetts Institute of Technology Artificial Intelligence Laboratory
Department of Electrical Engineering and Computer Science
545 Technology Square, Cambridge Massachusetts 02139

Rodney A. Brooks

Massachusetts Institute of Technology Artificial Intelligence Laboratory
Department of Electrical Engineering and Computer Science
545 Technology Square, Cambridge Massachusetts 02139

ABSTRACT

In the design and construction of mobile robots vision has always been one of the most potentially useful sensory systems. In practice however, it has also become the most difficult to successfully implement. At the MIT Mobile Robotics (Mobot) Lab we have designed a small, light, cheap, and low power Mobot Vision System that can be used to guide a mobile robot in a constrained environment. The target environment is the surface of Mars, although we believe the system should be applicable to other conditions as well. It is our belief that the constraints of the Martian environment will allow the implementation of a system that provides vision based guidance to a small mobile rover.

The purpose of this vision system is to process realtime visual input and provide as output information about the relative location of safe and unsafe areas for the robot to go. It might additionally provide some tracking of a small number of interesting features, for example the lander or large rocks (for scientific sampling). The system we have built was designed to be self contained. It has its own camera and on board processing unit. It draws a small amount of power and exchanges a very small amount of information with the host robot. The project has two parts, first the construction of a hardware platform, and second the implementation of a successful vision algorithm.

For the first part of the project, which is complete, we have built a small self contained vision system. It employs a cheap but fast general purpose microcontroller (a 68332) connected to a Charge Coupled Device (CCD). The CCD provides the CPU with a continuous series of medium resolution gray-scale images (64 by 48 pixels with 256 gray levels at 10-15 frames a second). In order to accommodate our goals of low power, light weight, and small size we are bypassing the traditional NTSC video and using a purely digital solution. As the frames are captured any desired algorithm can then be implemented on the microcontroller to extract the desired information from the images and communicate it to the host robot. Additionally, conventional optics are typically oversized for this application so we have been experimenting with aspheric lenses, pinholes lenses, and lens sets.

As to the second half of the project, it is our hypothesis that a simple vision algorithm does not require huge amounts of computation and that goals such as constructing a complete three dimensional map of the environment are difficult, wasteful, and possibly unreachable. We believe that the nature of the environment can provide enough constraints to allow us to extract the desired information with a minimum of computation. It is also our belief that biological systems reflect an advanced form of this. They also employ constant factors in the environment to extract what information is relevant to the organism.

We believe that it is possible to construct a useful real world outdoor vision system with a small computational engine. This will be made feasible by an understanding of what information it is desirable to extract from the environment for a given task, and of an analysis of the constraints imposed by the environment. In order to verify this hypothesis and to facilitate vision experiments we have build a small wheeled robot named Gopher, equipped with one of our vision systems.

Copyright 1993 by Andrew Gavin. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

1. PHILOSOPHY

In the design and construction of mobile robots vision has always been one of the most potentially useful sensory systems. However, it has also become in practice the most difficult to implement successfully. Here at the Mobot Lab we have designed a small, light, cheap, and low power vision system that can be used to guide a mobile robot in a constrained environment. At this point we are using as our target environment the surface of Mars. It is our belief that the constraints of this environment will allow the implementation of a system that provides vision based guidance to a small mobile Martian rover.

For many animals vision is a very important sensory system. In primates (and therefore humans) vision processing is the primary sensory mode and occupies a large portion of the neo-cortex. While it is clear that a variety of senses are essential to a mobile entity, be it robot or animal, vision has a number of substantial advantages. It is able to provide a survey of a fairly broad section of the world (approximately 200 degrees in humans) and at some considerable distance. The regular interplay of light and surfaces in the world allow internal concepts such as color and texture to be extrapolated, making object recognition possible. Additionally, vision is a passive system, meaning that it does not require the emission of signals. No other mode of sensation has all these properties. Chemo-receptors (smell and taste) are inherently vague in directionality. Somatic (touch) input is by definition contacting the organism, and therefore provides no input from distant objects. The auditory system is probably the second most powerful mode, but in order to provide information about inanimate (and therefore silent) objects it must be an active emitter, like bat sonar. However, it is only underground and deep in the ocean that the environment is uninteresting in the visual range of the spectrum.

Despite this, a useful artificial vision system has turned out to be very difficult to implement. Perhaps this is because the complexity and the usefulness of the system are linked. Perhaps it is also because no other system must deal with such a large amount of input data. Vision receives what is essentially a three dimensional matrix as input, two spatial dimensions and one temporal dimension. This input's relationship to the world is that of a distorted projection, and it moves around rapidly and unpredictably in response to a large number of degrees of freedom. Eye position, head position, body position, movement of object in the environment just to name a few. The job of the vision system is to take this huge amount of input and construct some small meaningful extraction from it. Nevertheless, whatever the reason for the difficulty in implementation, it is clear from ourselves and other animals that vision is a useful and viable sense.

For this project we had as our goal the construction of a small vision system that takes as a visual input the view from atop a small Martian rover. Its job would be to then quickly process this input in realtime and provide as output a small bandwidth of information reporting on the relative location of safe and unsafe areas for the robot to go. It might additionally provide some tracking of a small number of features "interesting" to the rover, for example the lander. The vision system was designed to be self contained. It has a pan and tilt camera head and an on board processing unit. It also draws a small amount power and exchanges a very limited degree of information with the host robot. The project has two parts, first the construction of a hardware platform, and second the implementation of a successful vision algorithm.

Professor Rodney A. Brooks, the head of the Mobot lab, has long been the champion of the belief that small cheap systems with a biologically based "behavioral" design can provide excellent results in real mobile robot applications [1]. He has demonstrated this with many small robots that have provided robust powerful performance with very small amounts of processing power. It is fairly widely believed that the Mobot lab's robots are some of the most successful fully autonomous robots built to date. They include notables such as Squirt [2], the tiny fully autonomous robot, and Ghengis and Atilla, a pair of highly robust small legged robots. Professor Brooks also believes that small cheap robots should be used in space exploration [3].

As to the second half of the project, it is our hypothesis that a simple vision algo-

rithm does not require huge amounts of computation. That goals such as constructing a complete three dimensional map of the environment are difficult, wasteful, and possibly unreachable. We believe that the nature of the environment can provide enough constraints to allow us to extract the desired information with a minimum of computation. It is also our belief that biological systems reflect an advanced form of this. They employ constant factors in the environment to extract what information is relevant to the organism.

This theory has already been used in our lab to implement a mobile robot that is "among the simplest, most effective, and best tested systems for vision-based navigation to date" [4; 5; 6]. We believe that these ideas can be combined with what is known about the Martian surface to create a system able to provide useful information to a Martian rover. The few existing Martian surface pictures returned from the Viking landers show a flat landscape of fine dust with protruding rocks.

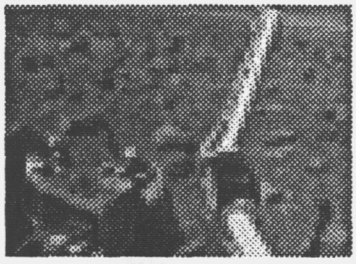
We have implemented several different algorithms on our system. Several of these attempt to extract the same navigational information from the scene. Each of these techniques is sensitive to different features, and so the simultaneous use of multiple approaches can yield added reliability. This algorithms are discussed below in detail.

We believe that it is possible to construct a useful real world outdoor vision system with a small computational engine. This will be made feasible by an understanding of what information it is desirable to extract from the environment for a given task, and of an analysis of the constraints imposed by the environment.

2. The Mobot Vision System: an actively controlled CCD camera and visual processing board

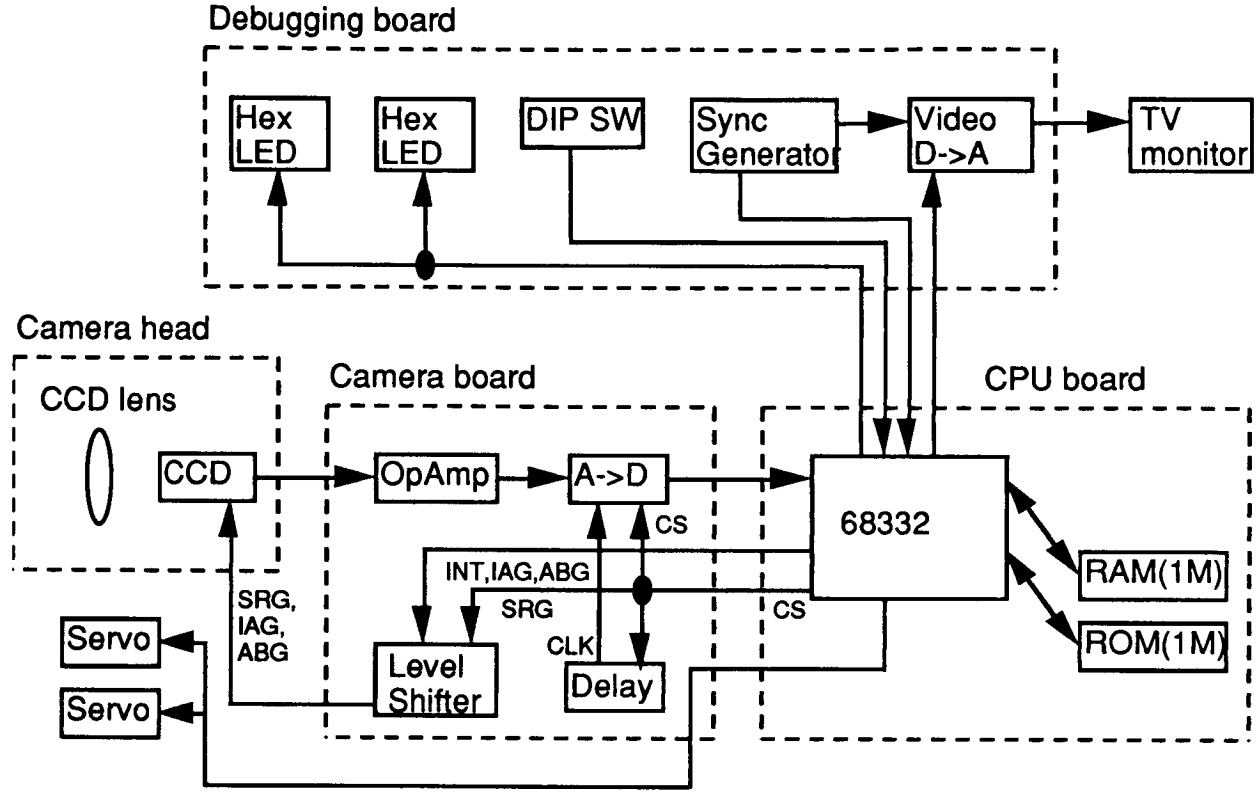
We had as our design goals the creation of a compact vision system which was cheap, simple, and had a low power consumption. We wanted to have everything needed for simple vision built in, including a significant amount of processing power. We chose the Motorola 68332 as the brain. This is an integrated microcontroller with on board RAM, serial hardware, time processing unit (TPU), and a peripheral interface unit. It has a decent amount of horsepower, essentially that of a 16-25 MHz 68020, and has a software controllable clock speed for power regulation. The 68332 was available in a simple one board solution from Vesta Technologies. To this we added 1 Megabyte each of RAM and of EPROM.

In accordance with our philosophy we decided on an image size of 64 by 48 with 256 levels of gray. These images are large enough that most important details of the view are clearly visible and recognizable to humans (see adjacent example image). Any number of gray shades in excess of about 64 are nearly indistinguishable, but 256 is convenient and allows for a neat single byte per pixel representation. Our chosen resolution requires an image size of 3K bytes. We also chose a frame rate of 10-30 frames per second. We believe that both the chosen frame rate and resolution are more than sufficient to allow the simple visual tasks which we intend this system to accomplish. They also make for a total bandwidth of 30-90K bytes/second, which is well within the amount of data that the 68332 can transfer while leaving lots of free processing time. It takes approximately 7 milliseconds for the processor to clock and read in a single frame. At 10 frames per second this only consumes 7% of a 16MHz 68332's processor time. This allows 93% for processing the images, which amounts to about 186,000 instructions per image, or about 62 instructions per pixel. 62 instructions per pixel, while it might not seem like much, or more than sufficient to do many of the simple vision tasks we have in mind. If more calculation is required, then a faster version of the chip can be used to double the available power.



To the CPU Board we added two piggy back boards: a Camera Board, and a Debugging Board. The Debugging Board has 2 hex LEDs for output, 6 bit DIP switch for input, a reset and debug button, and some video circuitry. These circuits are extremely

simple because the 68332's interface module allows nearly any kind of device to be mapped onto the bus. Even the video circuitry is very simple. We use two chips, a video level Digi-



Block diagram for the Mobot Vision System

tal to Analog Converter (Intech VDAC1800) and a video sync generator. The timing signals are run into the extra two input bits in our 8 bit input port and a trivial program on the CPU watches for transitions and outputs image data to the D to A convertor at the appropriate times. This allows us to display images contained in the CPU at 30 hertz to a video monitor with just two chips. Most of the work is done by the CPU. The system is capable of capturing images from the camera and outputting them to the video display at 30 hertz. Since video output consumes considerable CPU time it is not advisable to run the display at the same time as doing extensive vision calculations. However it provides a nice, easy output of the camera image in realtime which is essential for debugging, and for tuning the camera circuitry. There are also two switches on the Debugging Board to toggle the LED and video hardware for power consumption reasons. Additionally the system is completely capable of running without the Debugging Board attached, so that when the development phase is complete, it can be removed.

The Camera Board is the heart of the system. It contains a high speed Analog to Digital Converter (Sony CXD1175), some timing circuitry, the CCD level converters, and some output ports for our eye positioning servo actuators. It was one of the design considerations of this project that we should not use analog video at any point in our capture process (the video output on the Debugging Board is the only video circuitry in the entire system). At the resolution and frame rate we desire video adds unnecessary complications. It forces the use of high resolution CCD's, and a 30 hertz frame rate. Besides, since we desire to capture the frames digitally on the CPU Board there is no need to go through this convoluted intermediate stage. Instead we chose a low resolution CCD from Texas Instruments (TC211), clock the chip directly from the processor, amplify the signal, and read it through

the A to D converter straight back into the processor. A simple program on the processor generates the needed timings, and since the A to D converter is connected as a memory mapped device, it simply reads in the pixel data. Since the CCD is a 192 by 165 device the input program merely clocks over two out of three pixels and lines in order to subsample the image at 64 by 48. A slightly modified version of the software is capable of grabbing 192 by 144 at the same frame rate, but results in a consequent reduction of the number of instructions available to each pixel (about 7 at 10 hertz).

The generation of the timing turned out to be quite simple [7]. The Integration signal (INT) used to signal the exposure timing was simply connected to a CPU parallel pin, as was the Image Area Gate (IAG) signal used to clock lines downward on the CCD. A third signal, the Anti Blooming Gate (ABG) was generated automatically by the 68332's TPU at no cost in CPU time. The only difficult signal was the Serial Register Gate (SRG) signal which shifts pixels outs of the current row. This signal must be as fast as possible and timed precisely to the A to D converter's sampling clock in order to get the peak of each pixel's signal. Fortunately, since the 68332 automatically puts out a chip select signal to the A to D converter to signal its possession of the bus, we used this as the SRG. By running this chip select signal through an adjustable delay and then into the convertor's sampling clock we were able to match the time it takes the CCD and amplifier to actually output the pixel. The Camera Board has several adjustable potentiometers, an adjustable delay knob, a signal offset knob, and a signal gain knob. All must be adjusted together in order to achieve a good picture.

Also on the Camera Board is the level shifter circuitry used to drive the CCD chip. This was specially designed with both simplicity and low power consumption in mind. The CCD chip requires its clock signals to be at specific analog voltages and so we explored three methods of converting the processor's TTL level signals. The first method was to employ the driver chips sold by the CCD manufacturer for this purpose. We rejected this because of the high power consumption which seems to be unavoidable in high speed clock generation circuitry. The second method was to use an operational amplifier to add analog voltages. Because we wanted a low power circuit, and also wanted to reduce the number of components, we chose the third solution, which was to use analog switches that could toggle the voltage at a reasonably high frequency and which were fast enough for the processor's clock rate. Our circuit resulted in a total power consumption for the Camera Board of less than half a watt (when it is supplied +5V, +12V, and -12V).

From the Camera Board a six wire cable connects to the camera head. Since the robot needed to insure as wide an angle as possible, we explored small short-focal-length lenses. Generally wide angle lenses have several merits, such as a wider depth of focus, which makes a focusing mechanism unnecessary, a smaller F number, which increases the image brightness, and an insensitively to camera head vibration. However, it is sometimes difficult for wide angle lenses to compensate for the image aberrations. After testing several aspheric lenses, pinhole lenses, and CCD lens sets, we decided to use a f=3mm CCD lens from Chinon (0.6" in diameter and 0.65" long, 5g weight).

In front of the lens we placed ND filters in order to prevent over saturation of the CCD. Our CCD is actually quite sensitive and needs at least a 10% pass filter to operate in room level lighting, sometimes it even needs considerably more. In order to expand the dynamic range of the camera the frame grabbing software is designed to calculate the light level of the image and adjust the integration (exposure) time of the frame correspondingly. This adds an extra 10 decibels of dynamic range to the system, allowing it to work adequately in subjective indoor light levels ranging from lights off at night to sunlight streaming in through several large windows.

The camera is mounted on top of two Futuba model airplane servo actuators (FP-S5102). These are very small and light weight, and allow easy and fairly precise control of the camera position by the CPU. The servo actuators are connected via the Camera Board to the 68332's TPU. This allows the generation of their Pulse Width Modulated (PWM) control signals with virtually no CPU overhead. These actuators give the camera

both pan and tilt over a fairly wide range (170 degree pan, 90 degree tilt). The CPU has a number of simple routines that allow it to specify both absolute and relative positioning of the actuators, to read where they are, and preforms bounds checking to prevent the actuators from twisting the camera into themselves. The camera head and its servo actuators weigh 68 grams.

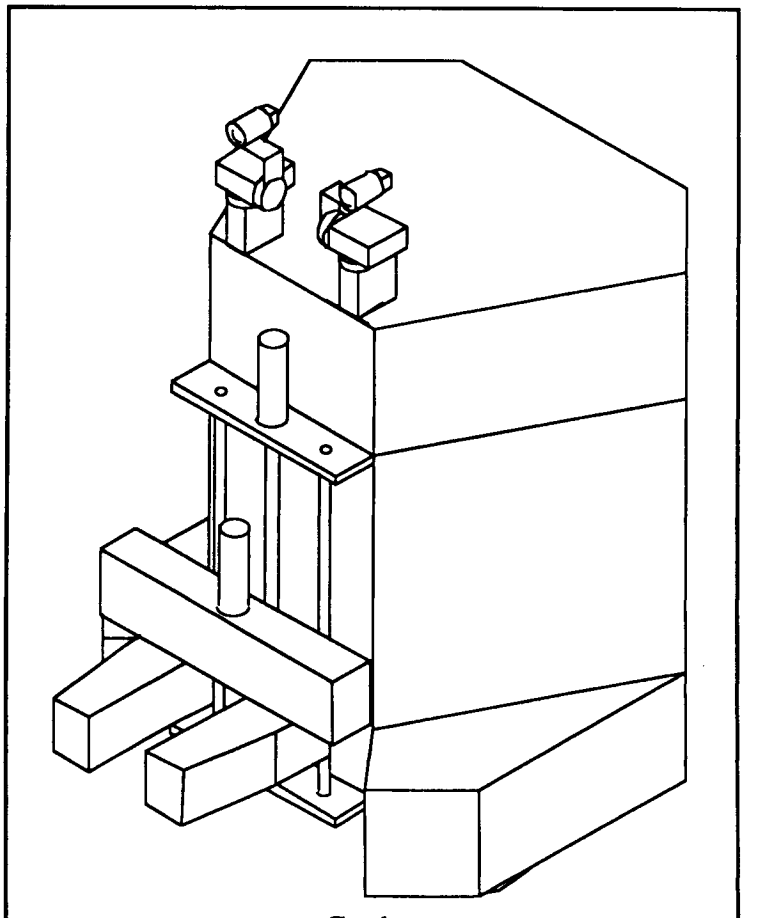
We have gone to a great deal of effort to minimize the power consumption of the Vision System and have been quite successful. The CPU board consumes 0.5 watts of power. The Camera Board also uses 0.5 watts. This means that the entire CPU and vision system consumes less than 1 watt of power. The video out circuitry on the Debugging Board requires an additional 1 watt of power, however there is a switch to disable this circuit, and since its use is for debugging this is not significant. The servo actuators also require some power. When idle they consume a mere 0.05 watts. Unless they are being moved constantly their average power draw is barely more than this.

Cost was also a significant factor in our design. The Vesta CPU Board costs $300. One megabyte of static RAM costs $200 (one could use less to save money, a megabyte of EPROM costs $25 (again one could use less to save money), two servos cost about $130, the CCD costs $35, the driver chips $30, the analog to digital convertor $25, the video chips $50, the power convertor $65, and miscellaneous other chips about $10. This is a total cost of around $700, many significant components of which could be eliminated to save money. One could use less RAM, or forego the servos or Debugging Board, possibly bringing a useful system as low as $350 in parts.

Overall this system is a small, cheap, and low power vision solution. It provides the input of 64 by 48 pixel 256 level gray scale frames at 10-30 hertz from a small camera with CPU controlled pan, tilt, and dynamic range, as well as about 62 680x0 instructions per pixel of processing time at 10 frames per second. All of the electronic circuits fit in a space 15 cubic inches big, consume less than a watt of power, and cost about $700 dollars to build. The available processing time is sufficient to do simple calculations such as blurs, edge detections, subtractions, the optical flow normal, thresholding and simple segmentation. A number of these can be combined to provide useful realtime vision based information to a robot.

3. GOPHER: A PROTOTYPE VISION BASED ROBOT

In order to fully test our system in a real environment we have been building a small vision based robot, Gopher (see diagram). This robot is based on a R2E robot from ISR. The robot is quite small (15 inches



Gopher

tall, 10 inches wide, and 12 inches long). It is wheeled, has a gripper on the front, and a number of sensors, including shaft encoders, touch sensors, and infrared sensors. These motors and sensors are controlled by a number of networked 6811 processors. A master 68332 processor, runs the behavior language which was designed here at the MIT Mobot lab. We have added a flux gate compass accurate to 1 degree and two of our 68332 based vision systems. The boards are all contained within the R2 case, which has been extended vertically for this purpose. We have mounted one of our CCD camera heads on its dual servo base on top of the robot, adding an extra 3 inches to the robot's height.

The Gopher robot provides a small and relatively simple low cost robot with an independently controlled camera and vision subsystem. We have used this system to implement many simple visual based tasks in a coordinated and integrated fashion.

4. Images: How good is 64 by 48

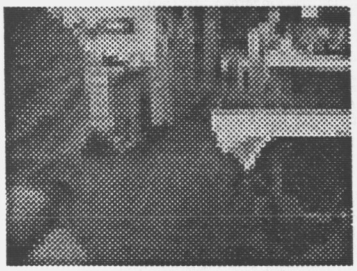
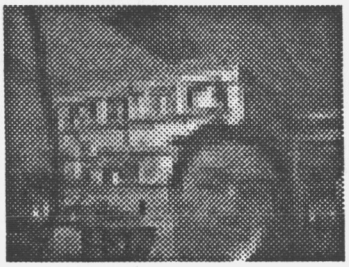
We have equipped the Vision System with a relatively wide angle lens (approximately 60 to 70 degrees). This is most useful for robot applications because the relative characteristics of space and objects around the robot are of more concern than the specific details at any one point.

Human perception of these images is quite good (see example images below). Objects approximately a meter square are easily visible at 20 feet. When angled down toward the ground the camera gives a good view of the space into which a forward driving robot will soon be entering.

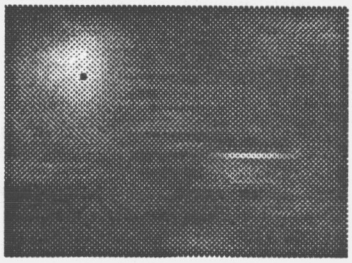
When all of the Vision Board variable knobs (adjustable delay, signal offset knob, and signal gain knob) are tuned properly the system captures an image with a full range of gray scales, which means a smooth clear image to the human eye. These parameters, while interrelated, can be tuned to a specific instance of the system in a few minutes. There is then little need to deal with them again unless a major system component (such as the amplifier, analog switch, or CCD) is exchanged.

The system is however fairly sensitive to light level. The CCD is very sensitive and requires at least a 10% pass filter to operate at normal light levels. In bright sunlight we usually add an additional 33% pass filter. By automatically adjusting the integration time in software we can cope with a moderate range of changing light levels, sufficient to encompass most operating conditions in an indoor environment. At some extremes of this range the image becomes more highly contrasting and less smooth. However this dynamic range is not sufficient to cover multiple environmental extremes, for example outside under sunlight and nighttime. To cope with this additional hardware would be required to increase the dynamic range. We have considered several other options. Filters could be changed as conditions vary, either manually, mechanically, electronically, or possibly using a kind of self adjusting filter (as found in some sunglasses). Additionally we are exploring the possibility of using a CCD with an electronic shutter. This would allow for a significant increase in the dynamic range, but would complicate the production of the CCD control signals.

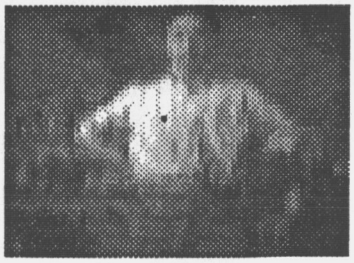
Example pictures from our system (64 x 48, 8 bit gray)

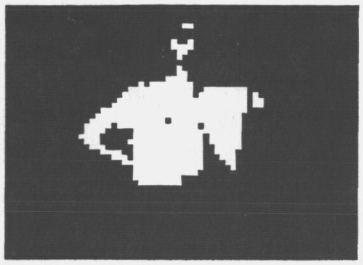
Blob Tracking



Black dot tracks the blob on this motion blurred image



Black dot marks the center of the segmented shirt



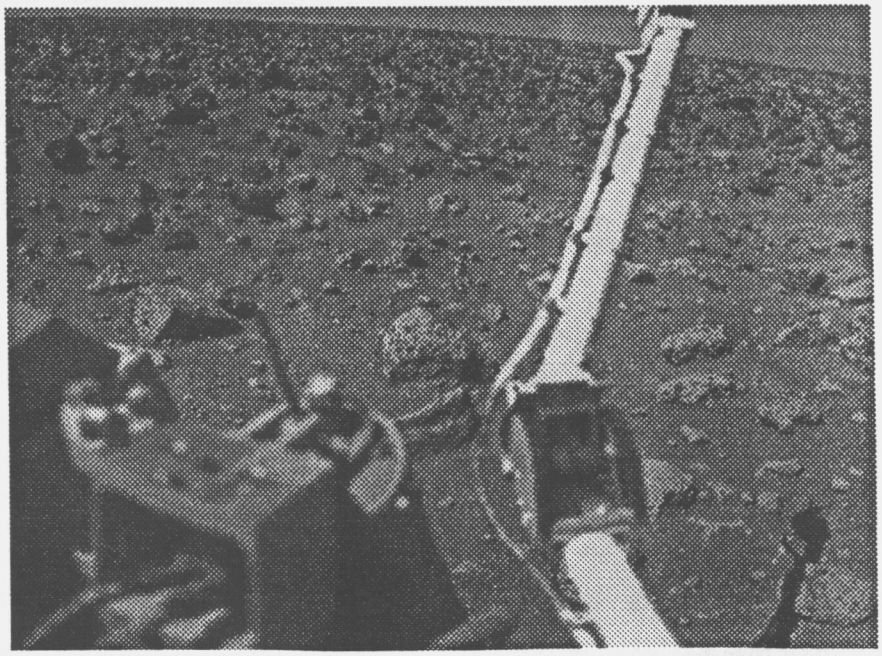
The shirt can be segmented by intensity

5. SOFTWARE

The Mobot Vision System is programmable in either 68332 (680x0) assembly or in C using the Gnu C Compiler (GCC). We have written a variety of basic routines. These setup the system, grab frames, actively adjust the integration time based on image light levels, output video frames, and move the camera via two servo actuators.

It takes approximately 7 milliseconds to grab a frame. This means that 10 frames per second occupies 7% of the CPU, leaving 62 assembly instructions per pixel free at this frame rate. We have coded in assembly a number of basic vision primitives. For a 64 by 48 8 bit image they have the following approximate costs: Blur (11 instructions per pixel), Center/Surround (11 instructions per pixel), Sobel edge detection (6 instruction per pixel), image difference (1 instruction per pixel), Threshold and find centroid (worst case 6 instructions per pixel).

As can be seen from the above figures a number of these basic operators can easily be combined to make a fairly sophisticated realtime (10 fps) visual algorithm. By making assumptions about the environment it is possible to construct algorithms that do useful work based on vision. For example, as a simple test case we have implemented code that thresholds an image and finds the center of any bright "blobs" in the image (see example images). This code requires at worst case 6 instructions per pixel plus some trivial constant overhead. We then use this information to center the camera on the "brightness center" of the image. The result is that the camera will actively track a person in a white shirt, or holding a flashlight. It is able to do so at 15 hertz while outputting video. This might not seem very useful, but by changing the thresholding conditions the camera would be able to track anything that can be segmented with a



Mars from the Viking Lander

small amount of processing. Intensity bands, the optical flow normal, thresholded edges, (and with filters) colored or infrared objects are all easy candidates for this technique.

We have used the Mobot Vision System to implement a host of other visual based behaviors. Here at MIT Ian Horswill has built Polly, a completely visually guided robot that uses an identical 64 by 48 gray scale image and a processor only somewhat more powerful than the 68332. Polly is designed to give "brain damaged tours of the lab." It is capable of following corridors, avoiding obstacles, recognizing places, and detecting the presence of people [5]. We have brought many of these skills to the Gopher system, and to Frankie, another lab robot based on the Mobot Vision System. The algorithms for avoiding obstacles and following corridors are easily within the power of one of these vision vision systems. Ian Horswill has also used one of these vision systems, slightly modified to add an additional camera, to do binocular gaze fixation.

6. Mobots in Space: Applications to Mars

NASA is planning on sending a small autonomous rover to Mars in the next few years as part of the Messur/Pathfinder mission. It is our belief that this rover could benefit from some vision based guidance, and that the approach used in the Mobot Vision System would be very suitable. A remote rover sent to Mars will be very limited by weight, size, and power consumption. The current rover design uses a digital CCD camera system quite similar to ours, as well as a low power processor. For power reasons the rover operates at very slow speeds, and so the algorithms we discussed here could run at a reasonable rate on its small processor.

The surface of mars as captured by the Viking Landers is fairly flat (at least where they landed) and regular (see Viking image above). It consists of a surface of tiny dust particles littered with an fairly Gaussian distribution of rocks. A rover's physical characteristics will determine what size of rocks are hazards and which are not. It would be useful for a vision system to be able to look forward and estimate roughly how much space in each direction there is until rocks that are too large to cross.

The first of our algorithms is texture based. This algorithm depends on a number of assumptions. These assumptions make it possible to extract useful data from an image in a reasonable time, however, it is important to be aware of the limitations they impose. If we assume that the ground is roughly flat, as indeed it is in the Viking images, then rocks that are farther away will be higher up in the image. Additionally the low resolution of the camera is convenient

A simple experiment employing a real Martian Image

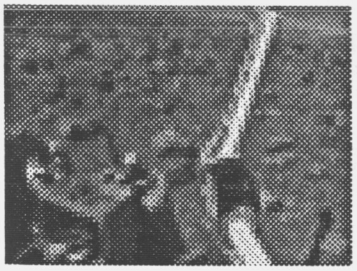


Image 1
(orginal)

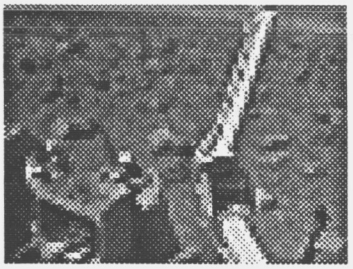
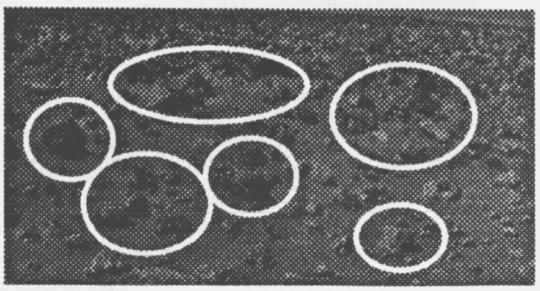


Image 2
(sharpen)

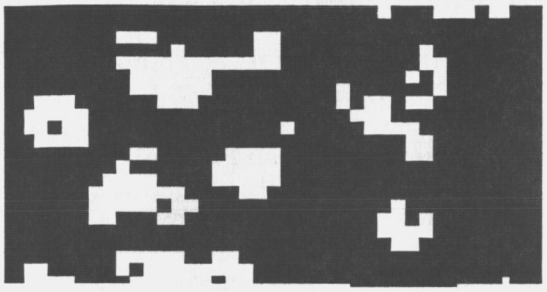
Image 3
(find edges)

Image 4
(threshold)

Close up of rock segmentation



High detail original, large rocks circled

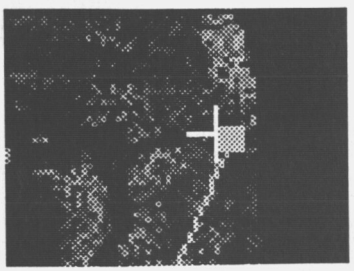


Segmentation of large rock clusters

because it will filter the dust and all small rocks into a uniform lack of texture. Texture, and therefore edges, is an indicator of objects or rocks. If the rocks can be segmented, then by starting at the bottom of the image and looking upward for "rocks" we can create a monotonic depth map of the distance to rocks in various directions. This is a simplified approach because it assumes that the ground is pretty flat. However it works surprising well on the Viking test image. Observe the series of images above. Number one is a 64 by 48 image as seen from the Viking Lander. Ignore the lander itself in the image, and assume a rover based camera can be mounted in the front where the rover itself will not be visible. In image 2 we have sharpened the edges of the images, and in image 3 we have used a simple edge detector. Finally image 4 was made by using an intensity threshold. Notice that the larger rocks in the original image are visibly segmented in the final image. By adjusting the threshold it is possible to segment for rocks of various sizes. This method is based on one devised by Horswill [4] and runs in realtime on the Mobot Vision System with processing to spare. We have used this system in several robots to avoid obstacles on flat uniform floors (usually carpets or cement). It works quite robustly.

The second algorithm is based on motion. We have implemented an algorithm that calculates the magnitude of the optical flow in the direction of the intensity gradient in realtime. If we make the assumption that the robot is moving forward roughly in the direction of the camera than the rate of motion of an obstacle is proportional to 1/d where d is the distance of the obstacle from the camera. This means that there will be very little movement in the center of the direction of travel, and more on the edges. Objects will accelerate rapidly as they approach the camera [8]. This large movement can be seen as increased flow, and with thresholding nearby obstacles can be loosely isolated. An even simpler strategy is to turn the robot in such a manner as to balance the flow on either side the robot. If there is more flow on the left go right, and vice versa. A large amount of flow in the

Optical Flow Examples



Light dots indicate flow. Arrow is preferred direction

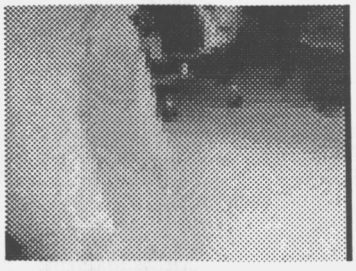
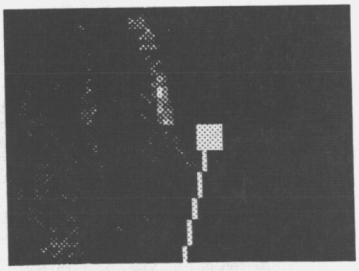


Image of an office with a person walking by



Flow of center image, arrow points away from flow

lower area of the image indicates an rapidly looming object. This technique is very similar to that employed by a number of flying insects. Balancing flow works well in a moving agent to avoid static objects. Some examples of this in action can be seen here. The line with the square on the end is an arrow indicating the direction the robot should go, the cross indicates an estimate of the direction of motion.

Because our algorithms run in real time, it is not necessary to convert to a complex three dimensional map of the world. We can convert straight from an image based map to a simple robot relative map. This is actually much more useful to a robot, and is vastly less computationally intensive. The realtime nature of our calculations applies a temporal smoothing to any errors made by the algorithm. This type of vision calculation tends to be very noisy, but with temporal smoothing, this noise is greatly reduced without have to resort to very difficult and time consuming calculations.

7. Conclusion

The images sampled by visual sensors, be they organic eyes, or inorganic cameras, contain a great deal of useful information about the environment, often available at more distance than other sensors allow. We believe that the limited amount of processing available on our Mobot Vision System, when combined with a low resolution image and reasonable assumptions about the environment will be sufficient to provide a robot with a good deal of useful information. We also believe that a small, light, cheap vision system such as this could be applied to many branches of the robotics field, including space exploration, indoor, and outdoor robots.

8. Acknowledgements

This research has been funded by the Jet Propulsion Laboratory Rover Technology Project, and we are grateful for their support. We would also like to thank Masaki Yamamoto, Ian Horswill, Olaf Bleck, Mark Torrance, Mike Connell, Dave Miller, and Anne Wright for their help.

9. References

[1] Rodney A. Brooks, "Intelligence Without Reason," *Computers and Thought*, IJCAI-91, 1991.

[2] Anita M. Flynn, Rodney A. Brooks, William M. Wells III, David S. Barrett, "Squirt: The Prototypical Mobile Robot for Autonomous Graduate Students," A. I. Memo 1120, MIT AI Lab, 1989.

[3] Rodney A. Brooks and Anita M. Flynn, "Fast, Cheap and Out of Control," AI Memo 1182, MIT AI Lab, 1989.

[4] Ian Horswill, "Analysis of Adaption and Environment," AIJ, 1992.

[5] Ian Horswill, "A simple, cheap, and robust visual navigation system," SAB92, 1992.

[6] Ian Horswill, "Characterizing Adaption by Constraint," *Proceedings 1991 European Conference on Artificial Life*, MIT Press, 1991.

[7] Area Array Image Sensor Products: Data Manual, Texas Instruments, Image Sensor Technology Center, 1992.

[8] Rodney A. Brooks, Anita M. Flynn and Thomas Marill, "Self Calibration of Motion and Stereo Vision for Mobile Robotics," 4th Int. Symposium on Robotics Research, 1987.

THE "MITY" MICRO-ROVER: SENSING, CONTROL, AND OPERATION

Eric Malafeew* and William Kaliardos†
The Massachusetts Institute of Technology and C. S. Draper Laboratory
Cambridge, Massachusetts

Abstract

The paper discusses the sensory, control, and operation systems of the "MITy" Mars micro-rover. Its compact and low-power sensor suite, with customized sun-tracker and laser rangefinder, provides dead reckoning and hazard detection in unstructured environments without aid from external sources. A high-level task architecture supports mapping, obstacle avoidance, GN&C, mission monitoring, and ground telemetry with high processing efficiency. For cluttered environments, reactive obstacle avoidance chooses the clearest trajectories with non-holonomic steering constraints and passing margin tradeoffs. Wireless operator interactions range from troubleshooting and reprogramming to graphical monitoring and supervisory control of the robot. The micro-rover system has been simulated in Monte-Carlo trials and field tested in various environments. Continuing work focuses on space qualification of the sensors and control software and further implementation of the ground station.

1. Background

1.1 The MITy Mission

MIT and Draper Lab-sponsored development of a low-cost Mars micro-rover began in 1990 and has since involved seventeen graduate and undergraduate students. This project supports the NASA MESUR objective of landing a micro-rover on Mars to scout and perform experiments on the environment. The "MITy" project goal has been to develop prototypes for this mission, which imposes strict constraints on size and mass, aid from external sources, and modes of operator interaction. The proposed operation scenario was to receive a set of destination commands from Earth daily, arrive at each to perform and record an experiment, then transmit the results back to Earth on the next cycle. Destinations would be chosen from rover video images and satellite maps. The current design scenario also allows supervisory monitoring and control for terrestrial and inner-space applications, in which communications are less limited.

Present contributions of the MITy project include studies on system requirements, design for mobility, soil-wheel interactions in low gravity, and most recently the sensory, control, and operation systems in this report [1,2,3,4,5]. Current team developments include system qualification procedures with JPL, vision for navigation and hazard detection, refinements of mobility and packaging, 3D simulation and animation, and robotic manipulator construction.

1.2 System Components

The second of three prototype platforms, MITy-2, is depicted in Figure 1. Its sensor subsystem is described in detail in Section 2.

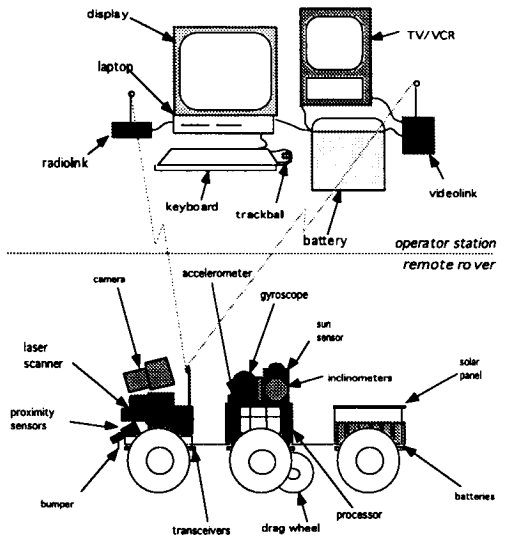


Figure 1: MITy System

Rover Breakdown

The micro-rover structure consists of three 15 x 15 cm articulated platforms, connected by a dual-spring suspension, that support sensing, processing, and payload subsystems. Six independently driven wheels enable the rover to climb steps up to 15 cm high and drive at speeds up to 30 cm/s on flat ground. The front and rear wheels are independently steered to permit a 63 cm minimum turn radius. Power is provided by a 30 amp-hr capacity battery, which can be recharged by a solar panel at a maximum rate of 6 W. The overall dimensions of the robot are 46 x 75 x 28 cm, and its total weight is about 9 kg.

*Graduate Student, Mechanical Engineering
†Graduate Student, Aerospace Engineering

Copyright © 1993 by Eric Malafeew and William Kaliardos. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

The processing subsystem consists of a Z-180 based micro-controller, with a throughput of about 12 Kflops/s and 512 Kb for code and data storage. Customized drivers generate motor and communications signals for a high level control interface. Software development supports "C" and remote debugging.

The current payload on the rover is a CCD camera, which sends images over a video transmitter to the operator station. Data communication occurs over a 9600 baud radiomodem with a LOS range of 2 miles. An optional LCD/keypad, not displayed, provides much of the operator station functionality for testing purposes.

Operator Station Breakdown

The main component of the operator station is a 486/66 PC-clone laptop, which runs the graphical user interface and programming environment. Operator devices are the computer display, TV/VCR, keyboard and trackball; communication devices are the radiomodem and video receiver. The small TV/VCR is useful for real-time supervisory control. These components are powered a 12 V car battery, which makes the entire operation station portable for testing and relocation.

1.3 Related Systems

Other notable Mars micro-rovers in development include JPL's Rocky, RATLER from Sandia, and Genghis from ISX Robotics, which differ largely in terms of system capabilities and realization. For instance, MITy is geared toward the extended MESUR mission beyond 10 m of the lander, while Rocky has been designed for operation within 10 m [6]. Larger rovers include CMU's Ambler and the Mars Rover from JPL.

2. Sensory System

2.1 Sensory Objectives

The sensor suite on the rover is required to provide sufficient navigation and hazard information to the control system for autonomous transit [1]. In particular, the subsystem requirements state that the rover position and attitude $\{x,y,z,\theta,\varphi,\zeta\}$ from the landing location should be determinable to within 10% of traveled distance. The hazard requirement states that the rover should be able to detect and localize potential hazards between the range of 1-10 ft, and within 1 ft be able to sense all types of accountable hazards. At the micro-rover scale these hazards include rocks, craters, and steep grades. Sensor selection, development, and placement are features of this problem.

Constraints that apply to the sensory system are to minimize power consumption, size and mass, and prototype cost. The sensory system should not rely on external sources and should be operable in the negligible atmosphere and magnetic field of space. Redundant sensors are also desirable in case of failures.

2.2 Navigation Sensors

By keeping within the 10% navigational error bound, the video image of the landscape can be compared with that from the previous day, reducing cumulative position error over multiple days. The video image plays an important role in navigation, since selection of the daily goals as well as position calibrations are based entirely on this data, as analyzed by the ground station [7]. The result is that the performance requirements of the rover navigation sensors are greatly relaxed, reducing cost, power, size, and complexity.

Dead reckoning is used to navigate to the stated accuracy, since this does not require an existing infrastructure such as land beacons or GPS satellites. Rather than using a costly inertial navigation package, the dead reckoning sensors are divided into three types: longitudinal translation, heading angle, and inclination.

Translation

Longitudinal translation is measured with the powered drive wheels of the rover, as well as an unpowered drag wheel. Drive wheel rotation is sensed with motor tachometers, which are supplied for speed control. An optical encoder is used to sense the drag wheel rotation. Both drag and drive wheel sensing is used to accommodate the potentially large variety of terrain. A drag wheel is beneficial when the powered wheels slip on soft surfaces, or when the vehicle wheels becomes partially unloaded due to rocks underneath its belly. However, since the drag wheel is smaller for packaging reasons it is less accurate than the drive wheels over rough terrains.

Heading Angle

Heading angle sensors include a sun sensor and an inertial angular rate sensor, hereafter called the gyro. The gyro is useful regardless of environment conditions, but at a cost of unbounded error growth with time. The sun sensor is used for dynamic calibration when shading is not an issue.

The selected rate gyro is a low-cost silicon vibrating beam sensor made primarily for automotive applications. The rate signal is integrated with a low-leakage analog circuit to provide a voltage signal that is

proportional to heading angle. Bias error in the gyro and integrating circuit is measured at stopping points for substraction in software.

The sun sensor was designed and constructed at CSDL, due to its unique requirements of a hemispherical field of view, small size, non-scanning, and low cost. To achieve these characteristics, a two-axis position sensitive detector (PSD) measures the position of a spot of light focused from a small fisheye lens. The position of this light spot is a function of the sun elevation (ε) and azimuth (β), as well as the inclination of the rover, as shown in Figure 2. Simple electronics are required to obtain light spot position from the PSD currents, making the entire sun sensor assembly a small rugged device that provides heading calibrations to within 0.5°.

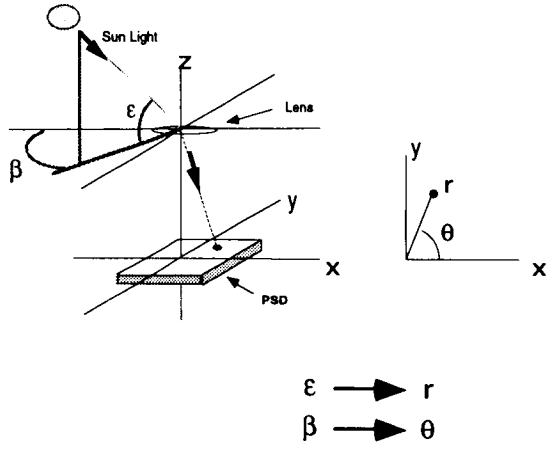


Figure 2: Sun Angle Sensor

Inclination

Two accelerometers provide tilt information for sun sensor calibration and propagation translation measurements out of the ground plane. These sensors are described in more detail below.

2.3 Hazard Sensors

One of the most difficult challenges of a micro-rover is sensing mobility hazards in an unstructured environment. The primary sensing problem is detecting objects for collision avoidance, although other mobility hazards exist.

Adequate sensing for a mobile robot typically requires a detailed depth map of the local terrain in all directions. The combination of high resolution and large field of view results in a large amount of data that requires much processing. Even if such processing were available, collecting the data with a small sensor is difficult. The MITy design presents a simplified hazard-avoidance sensor arrangement that meets its packaging requirements, and provides limited but sufficient sensing capability for autonomy. It is composed of a single axis scanning laser range finder, short distance proximity sensors, bump switches, and inclinometers.

Range Finder

Like the sun sensor, the range finder was designed and developed at CSDL in order to meet the unique needs of the micro-rover. For size and simplicity reasons, it works on active triangulation principles, illustrated in Figure 3. A solid-state laser produces a collimated beam in the near infrared. Light is diffusely reflected from the target, and a portion of this weak signal is collected by a small lens and focused to a spot on a 1-axis position sensitive detector (PSD), similar to that used on the sun sensor. Since the receiving lens is located a small distance from the laser, the angle of incidence of reflected light varies with range. Through triangulation, range can be calculated based on the reflected light angle, which is measured in the form of light spot displacement at the PSD.

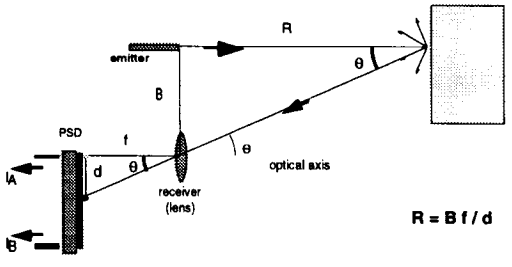


Figure 3: Triangulation Rangefinding

With simple electronics for transmitting and detecting, the complete range finder remained small, and yet ranges out to 3 m with 10% accuracy, which was specified from control system simulations [4]. The range finder is mounted to the front platform, and scans 180° in front of the rover at a height of 15 cm on flat ground. This plane-of-view approach is limited but provides adequate collision avoidance capability when used with other sensors.

Inclinometers

Sensing the tilt of a given rover platform is important for both navigation and hazard-avoidance reasons. The latter requires inclinometer data to calculate the orientation of the range finder beam, and to estimate the terrain geometry.

Accelerometers are used to sense the component of gravity when tilted, rather than the bulkier bubble level sensors. Miniature non-inertial grade accelerometers are

relatively small and faster, allowing pairs for pitch and roll on the front and middle platforms.

Proximity Sensors

Because of the planar range finding, vertical drops are sensed through two short-range proximity sensors. These are used on the front platform along with inclinometer data to estimate when the rover is partially over the edge of a cliff. The rover can then travel in reverse, using the high traction of the four rearward wheels.

Collision Sensors

Contact switches are located at the front and rear edge of the rover to detect collisions above a certain force. In addition, the odometry system can provide collision detection by observing front wheel speeds in response to commanded torques.

2.4 Implementation

The locations of individual sensors on the micro-rover are shown in Figure 1. All of these sensors are relatively low-performance, which allows them to be small, light, low-cost, and rugged. The total volume, mass, and power for the sensing system are about 2000 cm^3, 5 W, and 0.5 kg respectively, with a prototype cost of under $5 K. Rather than concentrating on fewer sensors that are high performance, this arrangement accomplishes similar tasks through sensor fusion in software, and additionally provides a certain degree of redundancy, increasing the system reliability.

3. Control System

3.1 Control Objectives

The semi-autonomous control problem is transit between a sequence of coordinate locations, without position or environment information from *a priori* or external sources. In addition, with sufficient communication rates, supervisory control should permit real-time operator interaction with sub-tasks, such as guidance and mapping. The robot should recognize traps that confuse its control logic, and be able to be recovered by the operator under supervisory mode.

Mobility, sensing, and processing constraints affect control system design. Mobility constraints include the minimum turn radius, overall dimensions, and climbing ability. Range, accuracy, and throughput of sensors must be recognized by the design, as well as processing throughput and memory.

3.2 Functional Breakdown

This control objective precipitates a variety of concurrent tasks, including mapping and obstacle avoidance, GN&C, mission monitoring, and ground telemetry. A functional breakdown of the autonomous transit mission is depicted in Figure 4. Boxed *tasks* perform an atomic function, which improves modularity and lessens code redundancy, utilizing other tasks and circled *resources*. *Map* and *Avoidance* tasks are related to trajectory generation. GN&C encompasses the *Guidance*, *Nav*, *Speed*, and *Steer* tasks. Lastly, *Sequence*, *Collision*, *Failure*, and *User* tasks comprise mission monitoring. The diagram also shows information transfer between tasks and resources. Task groups and their implementations are described below.

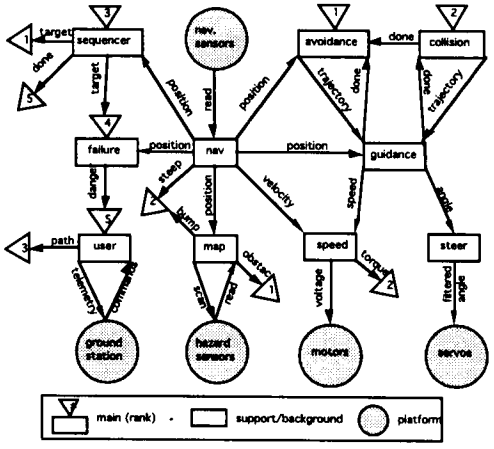


Figure 4: Functional Architecture

3.3 Task Architecture

The task architecture determines the scheduling of tasks and communications between them. The MITy architecture was mostly influenced by Payton's reflexive control ideas, which were implemented on DARPA's ALV [8]. Tasks are divided functionally and by cycle rates. Ideally each task would run concurrently, but this requires customized low-level scheduling and interprocess communications on a single processor. Instead, tasks are broken down into fast executing steps, which are interleaved by the *task planner* according to their intervals and priorities. The three types of tasks are *main*, *support*, and *background*; the type determines how a task is handled by the planner. Main tasks compete for motion control, support tasks aid main tasks, and background tasks aid all tasks. Information is communicated between tasks through a global variable pool.

All tasks are divided into *perceptions* and *reactions*. The planner decides which reactions to run based on the

"truth" of their perceptions. Any condition may exist in a perception, but every task has a *flag* and *interval*, which are set by the user to determine if and how often a reaction should be called. Background tasks have only these two conditions; support tasks will run only if their associated main task is running. Only one main task may run at a given time, which determines its *rank* as it executes. Main task perceptions have exclusive and positive *priorities* which are compared to the current reaction rank when true. A higher priority will subsume a lower rank, else the current reaction will continue without interruption. When a main task reaction ends without interruption, its rank is reduced to zero to allow the next highest priority main task to run.

This high level method is more efficient for the particular task designs than common real-time schedulers, which interleave tasks at the operating system level. The MITy task planner supports both pre-emption and concurrent execution at step resolutions, and does so without the overhead required for low-level context switching. Also, schedulers are not standardized and often unavailable for many processing platforms, which would lessen portability of the control code to future robots and simulations.

3.4 Trajectory Generation

Trajectory generation tasks produce heading and speed commands that maneuver the robot around obstacles toward the current target.

Mapping

The mapping function is a support task for obstacle avoidance. It sweeps the laser scanner 180° in 10° intervals, sampling the rangefinder at each stop. A reading is considered valid if its intensity is sufficient and the laser is not aimed at the ground. The last two sweeps of data are stored in a circular list, whose elements correspond to particular scanning directions. Elements consist of the coordinate endpoint of the last valid laser reading in that direction. This storage method constantly refreshes the obstacle list while maintaining complete coverage for asynchronous obstacle avoidance.

Obstacle Avoidance

The obstacle avoidance routine makes intelligent use of robot-centered laser information to maneuver through cluttered environments. It is the lowest priority main task--it is active in the absence of emergencies and consumes available processing time between other tasks. The general philosophy is borrowed from the VFH method [9], which is reactive in nature in that it does not plan ahead and must cycle quickly relative to robot motion. It has been shown more stable and predictable than methods based on artificial potential fields.

In the MITy approach, represented in Figure 5, polar map creation and trajectory selection are quite different from VFH. The independent axis of the polar map represents trajectory headings tangent to the minimum turn radius of the robot, while the dependent axis shows the "free distance," which is initially calculated as the distance to the first collision in all directions. The robot model is a circle that includes both its physical dimensions and the obstacle uncertainty due to laser and scanner accuracy. Free distance is then linearly traded off with safety, target heading error, and momentum heading error to obtain the weighted polar map. Safety is defined as the minimum distance to an obstacle in passing, which tends to guide the robot through the centers of clearances rather than narrowly on a side. The best trajectory is chosen in the direction of the highest free distance after weighting.

In constrast, the VFH method considers vehicle size at only one range, does not contend with steering constraints, and does not trade off safety and target error with free distance. It is therefore better suited to omnidirectional vehicles operating with a short range of concern.

The routine uses prediction and error correction to keep the robot in motion between trajectory updates. The nominal radius of concern and trajectory speed are respectively 3 m and 8 cm/s. If the robot sees no way out of a situation by moving forward, it produces a dead-end signal for collision recovery. In proximity to the target, the search radius is limited to prevent avoidance of obstacles behind it.

3.5 GN&C Module

Guidance, navigation, and control tasks concertedly command the drive and steering motors to follow a given trajectory.

Guidance

The guidance routine commands steering angles to meet a desired heading, using a proportional filter on heading error. It also stops the robot if it has not received a trajectory update after a fixed distance. Guidance supports both obstacle avoidance and collision recovery, depending on which produces trajectory commands.

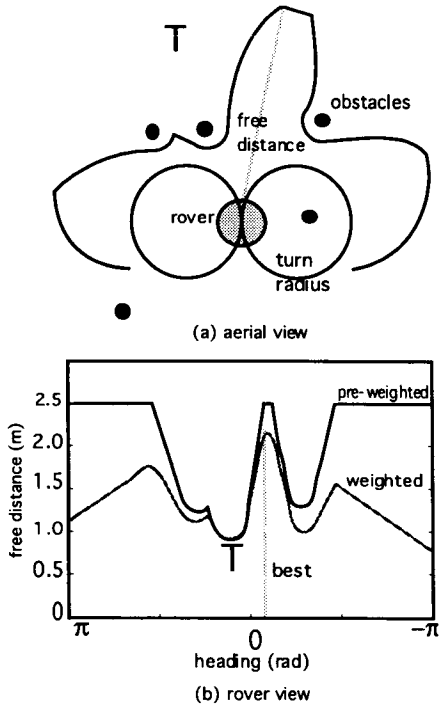


Figure 5: Obstacle Avoidance

Navigation

Navigation integrates heading, rate, and inclination information to update $\{x,y,z,\theta,\varphi,\zeta\}$ of the rover. Redundant sensors are arbitrated rather than averaged; the primary sensor is used unless disqualified, either by its own or other sensor readings. The sun sensor outweighs the gyroscope with a non-zenith sun, and the drag wheel outweighs tachometers over smooth terrain.

Control

Drive and steering control loops are separate but cooperative background tasks. Speed control uses tachometer readings and an anti-windup integral filter to generate motor voltages. Voltage commands to individual wheels are offset in accordance with steering rates and angles, to help aid steering and minimize slippage. Prohibitive torques, calculated from applied voltage and measured speed, are reported as collisions. The servo loop fixes steering and scanning rates and bounds to prevent power surges and motor damage.

3.6 Mission Monitoring

Mission monitoring tasks react to hazard collisions, monitor targets for completion or failure, and report progress to the ground station.

Collision Recovery

Rough terrain and reflective obstacles can cause the laser rangefinder to fail and may result in obstacle collisions. Collisions types are actual bumps, steep grades, deep craters, stuck wheels, and dead ends. When these occur, the collision task overrides obstacle avoidance to back the rover along its entry path, using a "look-ahead" path following method based on [10]. It then notes the collision location in the obstacle map for continued obstacle avoidance. If a collision is detected in reverse, the rover stops without updating the map and continues forward immediately.

Target Monitoring

Two functions are performed by target monitoring: sequencing and failure detection. When the rover reaches a target, determined by an estimated passing distance less than the minimum turn radius, the sequencing task overrides both collision and obstacle avoidance to perform a desired experiment. It then advances the target counter until the last target is reached. A target is considered failed if the rover does not advance on it or escape a given radius after a given amount of travel, as prescribed by [11]. Failure results in mission stoppage to conserve energy until operator intervention.

Telemetry

The telemetry function can operate in either the background or foreground. In background mode, it periodically sends position, obstacle, and free distance updates to the operator station, while servicing non-destructive sensor sampling requests and modifications to targets and parameters. Alternatively, the user may override semi-autonomous mode to set trajectories directly in supervisory mode, in which case telemetry is the highest priority foreground task.

3.7 Implementation

The control code was developed in standard "C." It is easily ported between the robot and simulation (described in Section 5) by replacing stump I/O functions.

Control parameters were initially determined from Monte-Carlo simulations and fine tuned on the actual robot. Obstacle avoidance throughput was traded off with background task cycle rates and obstacle mapping resolution and memory. As a result, most background tasks cycle at 10 Hz, while obstacle avoidance repeats at 1 Hz.

4. Operator Station

4.1 Operation Objectives

The capabilities of the operation station should include graphical monitoring and intervention for supervisory control, on-the-fly troubleshooting and reprogramming for semi-autonomous control, and data logging and replay for post-mission analysis of the rover. Communication bandwidth, time delay, interference, and range are restrictions on operation effectiveness.

4.2 Real-Time Display

A typical operator screen is shown in Figure 6, composed of a graphical window and text interaction areas. The text buttons emulate the optional keypad interface to the robot.

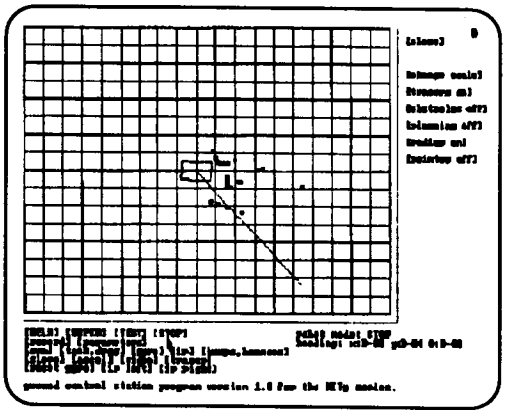


Figure 6: Ground Station GUI

A graphical window displays robot telemetry in real-time and allows user interaction with goal points. Robot position, obstacles, and free distances are updated independently at about 1 Hz onto a reference grid. These can be toggled for display as well as various driving aids for supervisory control. The scale and offset of the window relative to the robot can be adjusted on-the-fly. The screen follows robot motion by means of a travel boundary, which recenters the screen on the robot when crossed.

The button interface provides all means of interaction with the robot, from parameter and goal changes to sensor sampling requests. It also backs up graphical interactions, such as goal positioning and window resizing, with precise entry ability. A message window displays robot responses to user interactions and shows the precise navigation state at all times.

4.3 User Intervention

The operator interacts with the robot in one of three operation modes: semi-autonomous, supervisory, ready. Human factor issues in monitoring and shared control are regarded, as introduced by Sheridan [12]. The robot is placed into ready mode on power-up.

Ready Mode

In this mode the robot can service troubleshooting and reprogramming requests by the operator; all control parameters, goals, and sensors are accessible for viewing and modification. The drive system is inactive in this mode for safety reasons. Parameter sets may be saved and loaded from the operator station for testing or optimization purposes. The robot's initial position and path are set in this mode based on static video imagery; the mouse can set target destinations graphically. A panic button returns the robot to ready mode from other active modes.

Supervisory Mode

From supervisory mode the operator can command speed and steering to the robot. Arrow keys drive the robot forward or backward from 0-30 cm/s and can steer down to 63 cm arcs. Graphical aids for supervisory control include superimposed turning arcs, lines between targets, and unerased telemetry data. This mode is used for fine maneuvering or trap extraction, often relying on real-time video imagery in addition to position and obstacle telemetry. Although obstacle avoidance is inactive on the robot, it will still stop for collisions in case of operator error. To push an object or climb out of craters, collision parameters can first be modified in ready mode.

Semi-Autonomous Mode

In its baseline mode, the micro-rover autonomously travels between target destinations, which are initialized in ready mode. The operator may guide the robot in real-time by moving the current target with arrow keys, which move it radially and axially about the robot. The rover performs homing and obstacle avoidance at its nominal speed, freeing the operator to interact only when required. Real-time video can provide long-range information to the operator for maneuvering the rover around hazardous regions, rather than individual hazards.

4.4 Analysis Tools

Post-mission analysis involves logging and replaying telemetry data at a desired rate and display perspective. Telemetry signals may be recorded to a file from any mode during the mission. At operator station

startup, either real-time or replay operation may be chosen. The latter allows the user to load a telemetry file and analyze it with the same graphical aids available to the real-time mode, even if those aids were not active during recording.

4.5 Implementation

The operator station code is currently written in BASIC. A transmission delay of 0.2 sec was found by comparing the speed of implementing a simple command from the remote operator station to that from the optional keypad on the robot. This delay consists of both protocol overhead and radiomodem throughput; it is minimized by customized asynchronous and buffered message passing.

5. Performance Results

The MITy system has been implemented and tested in both the field and simulation. The simulation is used for development and Monte-Carlo performance evaluations.

5.1 Simulation

The rover simulation embodies the control code and two dimensional models of the platform and environment. It runs on an IBM/320 workstation at about 100 times actual rover speed.

Modeling

The vehicle, sensors, and environment models are low-fidelity, two dimensional representations intended for control system development. The vehicle is modeled kinematically as a bicycle with first-order steering dynamics. Perfect traction and rigid body assumptions are made at nominal speed, which has been shown reasonable for the micro-rover in low gravity packed powder environments [2]. The laser scanner and inclinometer models also have first-order dynamics; the drive motors, platform suspension, and other sensors are quasi-static at the time scale of interest. Rover dimensions and locations of sensors are represented faithfully. The laser is modeled as a ray and the bumpers as line segments in an environment of circular obstacles. All time constants and kinematic parameters were determined from experimental data, and sensor returns are considered ideal.

Monte-Carlo Statistics

Performance statistics were compiled on batch runs of the rover through random obstacle fields. In the nominal run, the rover must travel to a target 46 m (50 yd) away through 25 cm obstacles with 6% aerial density, the median distribution of rocks on Mars [13]. Runs were arbitrarily considered failures if the rover traveled over 92 m or crossed outside the 31 x 61 m field boundaries.

Measures of rover behavior and performance were selected by inspection and correlation studies, which describe time and power usage, path features, overall safety and navigational error, and failure modes of each run [4]. For a batches of 100 runs, statistics on these measures were compiled over variations in obstacle fields and control parameters; a few of these are listed in Table 1 for various obstacle sizes and densities. The mean passing distance is the average distance to the closest obstacle, which reflects rover safety. Normal deviation is of the distance away from the straight line path to the target, to estimate the area necessary to penetrate a given obstacle field. The total path distance shows power usage, while the distance in reverse indicates collision frequency. These measures were only compiled for successful runs, which is the foremost indicator of performance.

Table 1: Statistical Measures

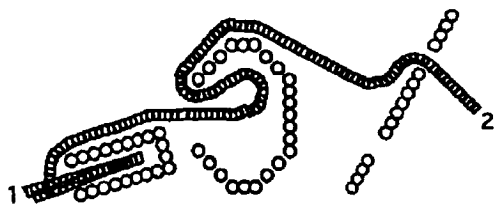
performance	obstacle radius (cm)/ aerial density (%)				
measures	25/ 6	25/ 10	25/ 3	30/ 6	20/ 6
mean pass (m)	0.8	0.7	1.2	0.9	0.7
norm dev (m)	1.4	2.5	4.3	1.2	2.1
total dist (m)	59	94	50	54	77
rev dist (m)	2.5	14.2	0.4	1.3	8.4
success (#)	100	44	99	99	94

Case Studies

Individual case studies highlight special abilities of the MITy control system. A rover that can recover from dead-ends without operator intervention, as shown in Figure 7(a), is desirable for cluttered obstacle fields with limited sensor ranges. In 7(b), the rover escapes the type of large concave obstacle that often troubles potential field methods of obstacle avoidance. Lastly, methods that ignore turn radius constraints would suffer in the hole-in-the-wall test, which demonstrates the unification of target homing with obstacle avoidance in 7(c).

5.2 Field Tests

The rover and portable operator station were transported to various locales to test real system performance. The results presented here are qualitative and more telling of hardware performance than the simulation.



(a) dead end (b) concave obstacle (c) hole-in-wall

Figure 7: Simulated Case Studies

Effects of Environment

Navigation and hazard detection were affected by different environments as listed in Table 2. Only problems are noted; otherwise, the rover performed as expected. Specular reflection and misorientation of the laser increased the frequency of collisions. Navigation was generally over 95% accurate outdoors, even over sand and under shade, due to dynamic sensor arbitration.

Table 2: Environment Effects

Environment	Nav. problems	Hazard problems
Hallways	gyro drift	specular reflection at low incidences
Pavement w/ traffic cones	sun reflectance off building windows	specular reflection on edges of cones
Sandy beach w/ sand piles	wheel slippage on sand	specular reflection off polished sand
Rocky beach	wheels not in contact w/ ground	pitching/rolling of laser scanner
Grass field w/ people	no problems	no problems

Supervision Effectiveness

Overall system performance and robustness were increased by using the various rover modes in conjunction with each other, which eases user fatigue during semi-autonomous segments while allowing detailed supervisory operations. In fact, much of the control system was debugged using the operator station for the insight and flexibility it provides. Real-time video images were mainly useful for target homing rather than obstacle avoidance, because the spatial relationship between the rover and observed obstacles was not intuitive to the user.

6. Continuing Work

Future plans in sensing are to incorporate a quartz gyro and phase-locking to a modulated laser for more accuracy and less power. The operator station is currently being integrated with customized 3D simulation and animation packages for operator training and further system verification. Further team effort will hopefully culminate in space qualification of the final MITy prototype for consideration in the NASA MESUR mission.

7. Acknowledgments

The authors acknowledge the support of the MITy team and its technical advisor, David Kang, PhD. Special thanks to Matthew Fredette for implementing operator station and driver designs, and Anthony Lorusso for implementing laser and sun sensor electronics. Lastly they appreciate Dr. Joseph Shea and Dr. Thomas Sheridan at MIT for their academic advisement. This project is supported by the C. S. Draper Laboratory and MIT Space Grant.

8. References

[1] Schondorf, S., *Systems Engineering for a Mars Micro-Rover*, MSAE Thesis, MIT, June 1992.

[2] Gilbert, J., *Design of a Micro-Rover for a Moon/Mars Mission*, MSME Thesis, MIT, Dec. 1992.

[3] Ma, C., *Dynamics, Control, and System Simulation of a Planetary Micro-Rover*, MSME Thesis, MIT, June 1993.

[4] Malafeew, E., *An Autonomous Control System for a Planetary Micro-Rover*, MSME Thesis, MIT, June 1993.

[5] Kaliardos, W., *Sensors for Autonomous Navigation and Hazard Avoidance on a Planetary Micro-Rover*, MSAE Thesis, MIT, June 1993.

[6] Layman, W. and Matijevic, J., "Micro-Rover Technical Baseline: Highlights, and Design Philosophy," *JPL Interoffice Memorandum*, July 1993.

[7] Krafft, B. and Pien, H., "Terrain Reconstruction from Rover Images," presented at *AIAA SP&T Conference*, Huntsville, Sept. 1993.

[8] Payton, D., "An Architecture for Reflexive Autonomous Vehicle Control," *IEEE Trans Rob Auton*, 1986, pp. 1838-1845.

[9] Borenstein, J. and Koren, Y., "The Vector Field Histogram--Fast Obstacle Avoidance for Mobile Robots," *IEEE Trans Rob Auton*, 1991, pp. 278-288.

[10] Amidi, O. and Thorpe, C., "Integrated Mobile Robot Control," *SPIE Mobile Robots V*, 1990, pp. 504-523.

[11] Lim, W., "Self-Awareness in Mobile Robots," *SPIE Mobile Robots VI*, 1991, pp. 58-67.

[12] Sheridan, T., *Telerobotics, Automation, and Human Supervisory Control*, MIT Press, Cambridge, 1992.

[13] Christensen, P., "The Spatial Distribution of Rocks on Mars," *Icarus*, v.68, 1986, pp. 217-238.

SUPERVISED SPACE ROBOTS ARE NEEDED IN SPACE EXPLORATION

Jon D. Erickson*
National Aeronautics and Space Administration Lyndon B. Johnson Space Center
Houston, Texas 77058

Abstract

Recent studies of the types, numbers, and roles of robotic systems for use in human space exploration, including the First Lunar Outpost (FLO) mission, with a focus on planet surface systems are summarized in this paper. These high level systems engineering modeling and analysis activities have supported trade studies and development of preliminary requirements for intelligent systems, including supervised autonomous robotic systems. The analyses are summarized, results are presented, and conclusions and recommendations are made.

One conclusion is that space exploration will be "enabled" by the use of supervised intelligent systems on the planet surfaces. These intelligent systems include capabilities for control and monitoring of all elements, including supervised autonomous robotic systems. With the proper level of intelligent systems, the number and skills of humans on the planet surface will be determined predominantly by surface science and technology (not outpost) objectives and requirements.

Space robotics, especially those systems being developed to operate on planetary surfaces, can be considered a form of the emerging technology of field robotics on Earth. The solutions to the problems we will be solving to make the exploration of our solar system possible and practical will apply to the many critical problems we have that require operating in hazardous environments on Earth and to improving human productivity in many fields.

* Chief Scientist, Automation and Robotics Division, Member AIAA

Copyright © 1994 by the American Institute of Aeronautics and Astronautics, Inc. No copyright is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all right under the copyright claimed herein for Governmental purposes. All other rights are reserved by the copyright owner.

1. Introduction

Human space exploration is a strategy for stimulating the United States, its people, and its economy as much as it is a strategy for exploring the Moon and Mars. A White House report [1] has outlined various visions and architectures for this crucial effort. We take the position in this paper that the greatest benefit to the U.S. economy of any space-exploration-related technology can come from the development of supervised intelligent systems, including supervised autonomous robotic systems. Such systems are mandatory for space exploration [2] to improve safety, reliability, and productivity, while enabling large cost savings through minimizing logistics [3]. Such systems are also needed in the U.S. economy [4, 5].

Intelligence is the ability to acquire and apply knowledge and skills to achieve stated goals in the face of variations, difficulties, and complexities imposed by a dynamic environment having significant unpredictability. Intelligent systems are composed of sensors for sensing the "world," effectors for acting on the world, and computer hardware and hardware and software systems for connecting the sensors and effectors in which a part of the processing is symbolic (nonnumeric). This processing enables practical reasoning and behavior, which in humans we call intelligence.

Examples of artificial intelligence capabilities in intelligent systems are knowledge based systems, expert systems, natural language understanding systems, robotic visual perception systems, intelligent control and planning systems, qualitative and model-based reasoning systems, and supervised autonomous robots. Many supply an explanation facility that enables the user to ask what reasoning was used and why the conclusions were reached. Intelligent systems can be of four basic kinds: nonmobile, nonmanipulative systems such as monitoring and control systems; nonmobile, manipulative systems such as robot arms fixed in place at the shoulder; mobile, nonmanipulative systems such as inspection robots; and mobile, manipulative systems such as mobile robots with arms and end-effectors. While supercomputers, distributed computers, or parallel computers are

currently required to achieve real-time performance with large scale intelligent systems, CPU speeds double every 6 months, so such intelligent systems will be easier and cheaper to achieve in the future.

It is important to understand the advantages intelligent systems have over conventional automation. Some advantages are given by Erickson [6], which are primarily perception and flexibility in dealing with uncertainty and dynamics imposed by real environments.

The benefits of using intelligent systems in space missions are improved and increased safety, reliability, and productivity. These benefits are derived from applying more knowledge and reasoning in more flexible and appropriate ways than conventional automation.

The EVA helper/retriever effort [7] is an initial attempt to build and understand a limited version of a supervised autonomous robot for use in space. Many other efforts to build intelligent robotic systems, not necessarily for space, are under way [8].

2. Space Exploration Studies

Recent studies [9] of the types, numbers, and roles of robotic systems for use in the 20-year Option 5A space exploration mission [10], with a focus on planet surface systems, are summarized in this section. These studies employed high level systems engineering models that we developed. We now employ a software modeling tool, the mission simulation and analysis tool (MSAT) [11], which enables us to account for the nonlinear effects of resource allocation, parallel support and mission tasks, and occurrence of contingencies.

Mission feasibility is a paramount issue in requirements generation (along with verification, validation, and traceability). A useful device that exercises the skill and judgment of those concerned with requirements is to tell the story of the mission.

These stories form the basis of input to MSAT. Any mission story will be in the form of a process description. At the requirements stage, the story of any subprocess (such as landing on the surface, unloading, etc.) will be in terms of objects specified by functionality, not by actual design. As the stage progresses toward design, the stories will involve process designs and objects wherein performance and operational parameters can be quantified.

With the process description format, each mission story is told in terms of parallel processes, each with prescribed start times. Each process has a functionality type; at present the types used in MSAT are the following:

- Mission backbone (e.g., landing, launch, site preparation)
- Science
  - EVA: geologic traverses, astrophysics, geophysics
  - IVA: lab experiments, life sciences, analysis, packaging
- Maintenance
  - Dusting
  - Servicing
  - Repair (EVA, IVA)
  - Replacement
  - Testing
  - Inspection
- Logistics
- Support
  - Power
  - Thermal control
  - Communications
  - Crew safety and well-being

Each process is broken down into subprocesses, called stages, and each stage has a set of options corresponding to the different ways in which the stage can be carried out. Each stage option has a model assigned that enables computation of elapsed time versus stage option name and the types of agent resources to be used:

- EVA, IVA, and equipment
- EVA, robotics, and equipment
- IVA, robotics, and equipment
- EVA, IVA, robotics, and equipment
- Robotics only and equipment

MSAT is written in (interpreted) C, which is an application running under the Ellery Open System (EOS) [12]. EOS is a development and run-time environment for distributed computing applications. MSAT is a relational, table- and model-driven simulator that makes allowances for parallel processes and dependencies, for supply and demand of resources to accomplish processes, and for elapsed time in accomplishing mission processes and tasks.

In constructing the Option 5A models, we first reviewed the story of the mission from previous accounts that tells what is intended to be done during the mission with flight times, site layouts,

element and system descriptions, and manifests. Then we examined various advanced automation and robotics issues raised by the story. After establishing two differing points of view, a conventional systems view and an intelligent systems view (causing changes in the equipment manifests and in the way mission tasks are carried out), we redescribe the missions from those points of view (see Table 1) and construct two models corresponding to these views for the purpose of obtaining comparative results.

Numerical models were constructed only for control and monitoring, unloading, site survey and regolith handling, emplacement, servicing, and maintenance.

Figure 1 shows results from modeling crew workload demand for selected tasks versus crew EVA availability under the conventional systems model for four astronauts in 21 years of lunar missions. As can be seen, either more capable equipment, as in the intelligent systems model, or more crewmembers are required. This mission scenario, which calls for complex activities such as offloading of large equipment and construction of facilities in the absence of humans at the planet surface site, clearly makes intelligent robots mandatory. Figure 2 shows the crew EVA demand under the intelligent systems model and shows primary crew time for creative activities of exploration, science, and planetary resources use. Science and engineering skills in the crew may now replace some pilots and technicians. A supervised autonomous outpost is thus seen as mandatory to preserve small crew sizes and ambitious surface mission objectives.

A broad range of robotic system uses in Earth orbit or during space transport is indicated by current studies. These include assembly of very large spacecraft systems such as propulsion systems and aerobraking structures [14]. Maintenance of onboard equipment in Earth orbit or during space transport is another robotic system use being studied.

3. First Lunar Outpost Studies

This section is based on Erickson [15], which has more details. The JSC Automation and Robotics Division (A&RD) has been performing high level systems engineering modeling and analysis activities to support trade studies and systems effectiveness analyses for proposed missions to the Moon and Mars. Preliminary requirements for intelligent systems, including supervised intelligent robots, have been the focus of our efforts.

Table 1 – Conventional systems versus intelligent systems

• Use Fisher-Price[13] recommendations	• Use Fisher-Price recommendations
• Conventional software – DDBMS for knowledge representation – Normal sensors	• Intelligent system software – State-of-affairs knowledge representation – Extensive sensors/perception for knowledge acquisition – Ability to use knowledge
• Mainly surface teleoperation, limited telerobotics	• Supervised, autonomous robotics with structured environments
• Rudimentary, mainly Earth-based DOKSS	• Distributed DOKSS, real time where needed
• Ground-based control and monitoring (for Moon)	• Surface-based, built-in control and monitoring with ground-based oversight
• More-than-minimal computing power	• Major computing power and information storage on surface
• Predetermined procedures	• Adaptable procedures with built-in precautions – Rehearsals – Interelement and interface testing
• Limited surface diagnosis and repair	• Design for ease of testing, diagnosis, servicing, maintenance, and repair
• Limited surface communication, major downlink	• Major surface communication, major downlink
• Crew used for outpost operations and maintenance, science and technology deployment	• Crew used for science and technology, minimal outpost operations and maintenance

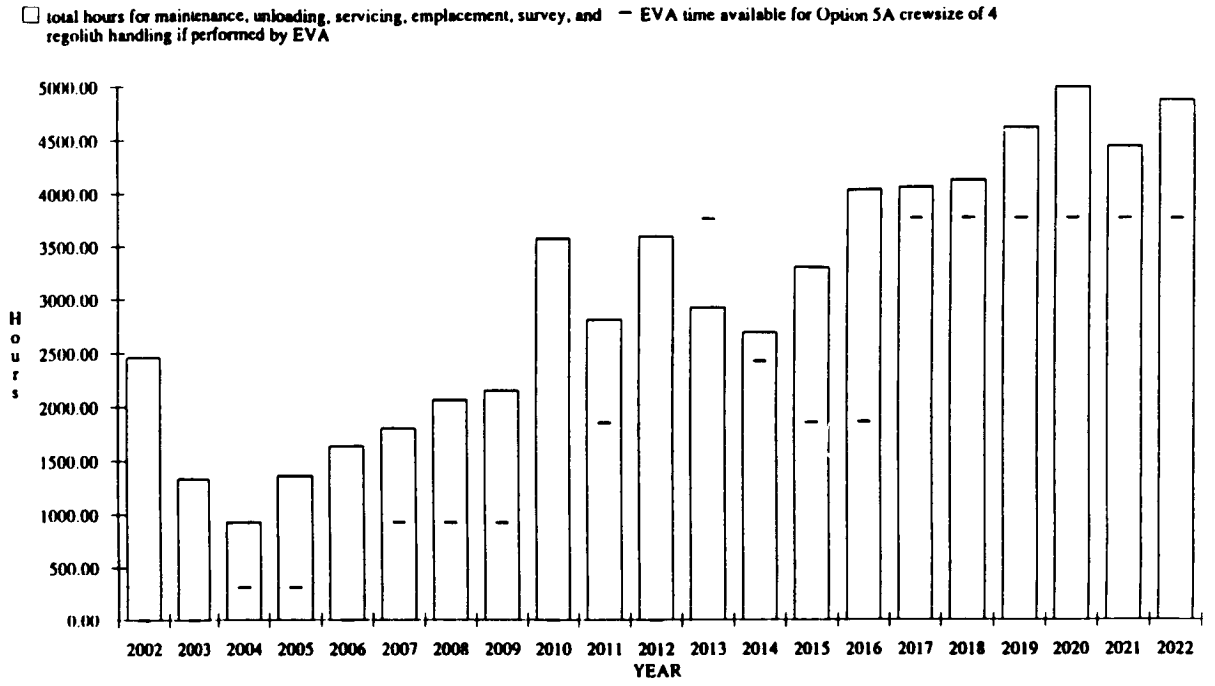


Figure 1 – EVA allocation for conventional systems approach.

Simulation is concerned with identifying and solving problems by testing how well the operation of engineered designs will meet the mission objectives. Simulation of operations can provide early identification of performance problems of integrated design and operations concepts. When applied with alternative process and equipment designs, simulation of operations is used to obtain a less costly short cycle run-break-fix [16] approach that can be iterated until simulations do not "break" anymore. Specialty engineering analyses, particularly reliability and maintainability, are most effective when implemented early in the design process when they can have the greatest

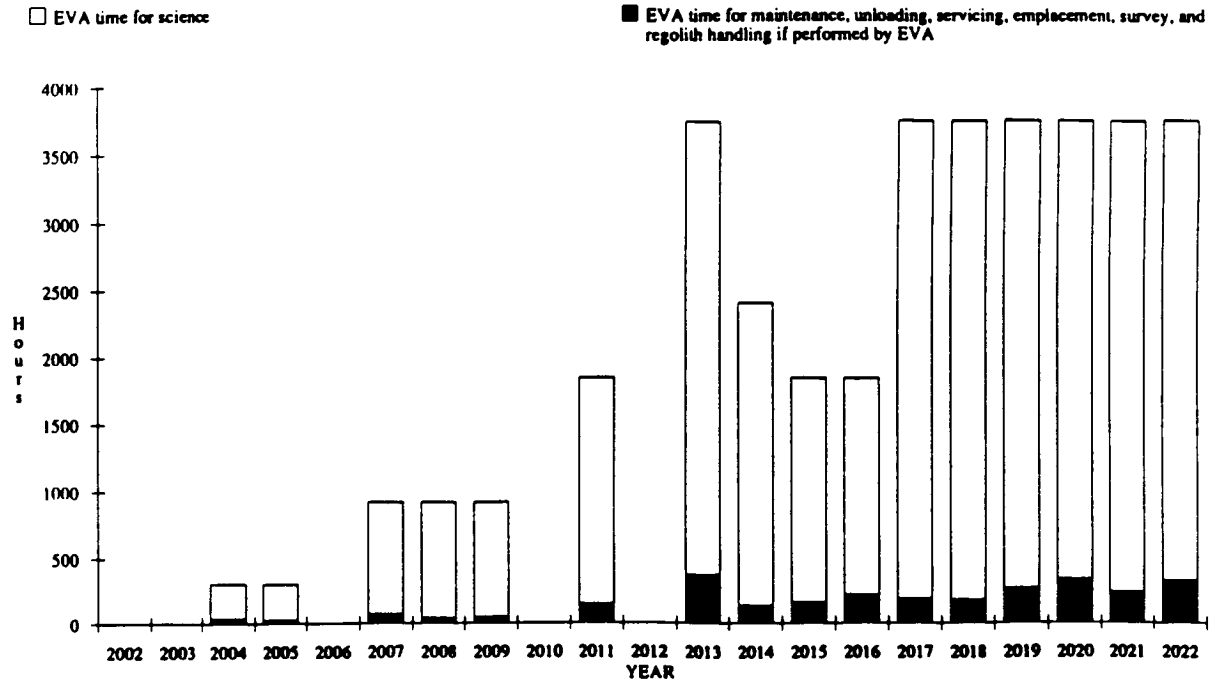


Figure 2 – EVA allocation for intelligent systems approach.

impact on overall design decisions. The JSC A&RD has developed MSAT for use in evaluating the feasibility and effectiveness of proposed mission concepts. We have also used a reliability and maintainability assessment tool (RMAT) developed by the JSC Reliability and Maintainability Division [17, 18] for SSF applications to estimate the amount of maintenance for the lunar surface habitation element.

The FLO mission, while being significantly more complex than any single Apollo flight, is vastly less complex than the Option 5A mission analyzed previously. Although there are periods when humans are not present, no offloading of equipment or construction of facilities is planned when crew is not present. Maintenance of facilities will still be required. We have continued to perform mission simulation and analysis to support system effectiveness studies of FLO and to understand the requirements for intelligent systems automation and robotics.

The FLO mission is envisioned as the first waypoint in expanding human presence in our solar system. FLO is established by the successful landing of an unoccupied human habitation element on the lunar surface and a subsequent 42-day visit by a crew of four that is transported to the lunar surface in a separate crew lander. The crew will arrive from 2 to 6 months after the cargo vehicle with the habitation element has landed. The habitat will be activated and checked out remotely before crew departure from Earth and will be in a ready state for crew arrival. Revisits to the outpost are projected at intervals of about 6 months. Humans are not present at the outpost during this interval; however, the outpost must be maintained sufficiently to allow reoccupancy.

During the 42-day (lunar day, night, day) FLO first mission, the crew will

- Perform equipment checkout and maintenance.
- Unload and transfer equipment and supplies between the crew lander and the habitat.
- Conduct local exploration and sample collection.
- Deploy scientific instrumentation (e.g., for space physics and astronomy).
- Deploy in situ resource utilization (ISRU) demonstration equipment.
- Conduct engineering and operations tests (e.g., human and equipment tests under varying and extreme thermal and illumination conditions).
- Perform life science experiments and IVA laboratory analyses.
- Perform crew self-sustenance and operational activities (e.g., housekeeping, training, planning, eating, resting, public affairs communications).

The habitation element provides all the facilities and subsystems (e.g., environmental control and life support, temperature and humidity control, data management) required to sustain the crew, except for food, personal items, and logistics resupplies that are transferred from the crew lander. The habitation element concept is an adaptation of the SSF habitation module with deployable solar panels, thermal radiator, and high-gain antenna. An airlock is provided for crew ingress and egress with provision for lunar dust abatement. Regenerative fuel cells provide power during the long lunar night.

3.1 Maintenance Simulation and Analysis

Maintenance has been investigated as a critical issue of the FLO mission. As a critical issue, maintenance or lack thereof impacts the following:

- Safety and survivability
- Mission goals
- Levels of performance
- Logistics and spares (and related mass and volume)
- Redundancies (and related mass and volume)
- Levels of commonality
- Designs for maintenance and repair
- Designs for diagnosis
- Control, monitoring, and fault diagnosis
- Tools and equipment
- Sensing and sensors
- Crew availability
- Amount and types of robotics
- Cost

The requirement addressed in analysis to date is to estimate the number of maintenance actions to be required as a function of time in the mission and the crew time required to accomplish the required maintenance. This will allow us to address the maintenance impact on the mission story as implemented in MSAT and those results. In

addition, it will give early insight into the alternative options and impacts to be considered.

A simulation tool for estimating maintenance demand has been developed for the SSF program and has been used for the FLO analysis reported here. This simulation tool is RMAT developed by the JSC Reliability and Maintainability Division and Loral Space Information Systems. The following discussion of this tool is paraphrased from Blumentritt[17] and the Assembly and Maintenance Implementation Definition Document[18].

RMAT is a stochastic, event-oriented simulation process written in Fortran and implemented on a personal computer. System maintenance is simulated at the individual component replaceable unit level of detail. Input to RMAT is a data base, which for each replaceable unit contains reliability data of the mean time between failure (MTBF), equipment reliability class (i.e., electronic, electrical, electromechanical, mechanical, structural, and structural-mechanical), and the life limits. Maintainability data includes the replaceable unit location (internal or external), mean time to replace (MTTR), mean time between preventive maintenance (MTBPM), and the number of crewmembers required for the maintenance. Robotic requirements can also be defined. Operations data in the data base includes the manifest and activation stages and the equipment duty cycles.

Factors that contribute to the generation of maintenance actions are the following:

- Random failures based on a lognormal distribution of the MTBF
- Early failures that are time-varying multipliers of the random failure rates and are based on a history of experience of spaceflights and satellites
- Preventive maintenance actions that are scheduled actions
- Life limit failures that are beyond the length of time of the current FLO study reported here

A Monte Carlo simulation approach is used to estimate failures. The duty cycle is a part of this calculation, as is a cold failure rate to account for failure rate contributions when equipment is not operating. K-factors[13] are applied as a failure rate multiplier to account for maintenance actions that occur for reasons other than the inherent component failure rate. For the FLO study, we used the default values that were developed by the SSF In-Flight Maintenance Working Group[18].

Maintenance time consists of work site time plus overhead time. Work site time is the time required to remove and replace the line replaceable unit (LRU) at the work site. Overhead time includes the time to get the replacement part and tools, travel to the work site, set up, close out the work site, and return parts and tools. A lognormal distribution is used to simulate the variability in the work site and overhead times. To estimate the amount of crew time required, maintenance actions are packaged into EVA and IVA crew shifts. SSF definitions were used: one IVA shift is composed of two crewmembers for 8 hr, each one performing 4 hr of maintenance; one EVA is composed of two crewmembers for 6 hr with 1 hr of sortie overhead.

In order to utilize RMAT to predict maintenance demands for FLO, a suitable reliability and maintainability data base was required. Since FLO was at the conceptual design stage, a representative data base was sufficient. The similarity of the FLO habitat elements and subsystems to the SSF habitation module and distributed subsystems suggested that SSF component reliability data can be used as a reasonable approximation for FLO habitat component reliability data. The SSF program developed a reliability and maintainability data base of predicted values for its own maintenance analyses[18]. We utilized that data (circa 1991) to build the FLO data base where elements were in "common" between FLO and SSF.

The mean work site time (MTTR) was estimated for each LRU and recorded in the data base. We used SSF times from the SSF data base, and where items were added we made separate estimates by comparison to SSF estimated times.

Overhead times can be input at the time of execution of RMAT. We used 0.5 hr for IVA overhead times. For EVA overhead times, we chose to perform a parametric analysis and used overhead times of 0.5 hr and 1.0 hr for each LRU. We view a mean overhead of 1.0 hr as an optimistic goal for FLO EVA maintenance actions.

3.1.1 Simulations and Results

Our approach to maintenance analysis is to perform parametric simulations that will provide answers (or insight into the answers) to key questions such as the following:

- What is the level of maintenance actions indicated?

- What is the crew demand time (work site plus overhead) to perform these maintenance actions?
- How many EVA and IVA shifts are required, and do they fit within the preliminary allocation?
- How does the number of maintenance actions vary when crew is present and is not present?
- What level of maintenance action backlogs exists?
- What is the effect of delays in crew arrival from 2 through 6 months after the habitat has landed?
- What is the maintenance load for follow-on outpost visits?
- What is the impact of backlogged maintenance on habitat functionality?

To answer these and other questions, we formulated two basic maintenance scenarios: (1) instantaneous replacement, which gives an estimate of the maintenance load (a maximum) required to maintain the habitat in a full-up operational capacity and (2) scheduled resources where maintenance is delayed until crew arrival, which gives backlog estimates and functionality impacts. Both scenarios assume 100 percent diagnosis of failures and no cascading failure effects. For each of the two scenarios, we simulate 2-, 4-, and 6-month delays of crew arrival. We estimate EVA, IVA, and total maintenance actions and use both 0.5 and 1.0 hr. for EVA maintenance action overheads. For each scenario, we also simulate two follow-up missions of 45 days at 6-month intervals after each crew departure back to Earth. For each simulation, 50 to 100 runs (more than sufficient) are made by RMAT to calculate the results.

We have also performed the simulations with and without the early failure model to establish the bounds on results. Although the early failure model is considered to overestimate the number of maintenance actions, it is considered the better estimator for planning purposes.

Figure 3 shows the bounds on cumulative maintenance actions with instantaneous replacement for the two cases: (1) all maintenance action (MA) types and (2) all MA types excluding early failures. We also show these for the two duty cycles – crew present and crew not present (standby mode). The failure rate for the standby duty cycle is 20 percent less than that for the duty cycle when crew is present. For most of the scenarios, however, the FLO is in standby mode for a greater period of time than with crew present; therefore, the cumulative error rate will be closer

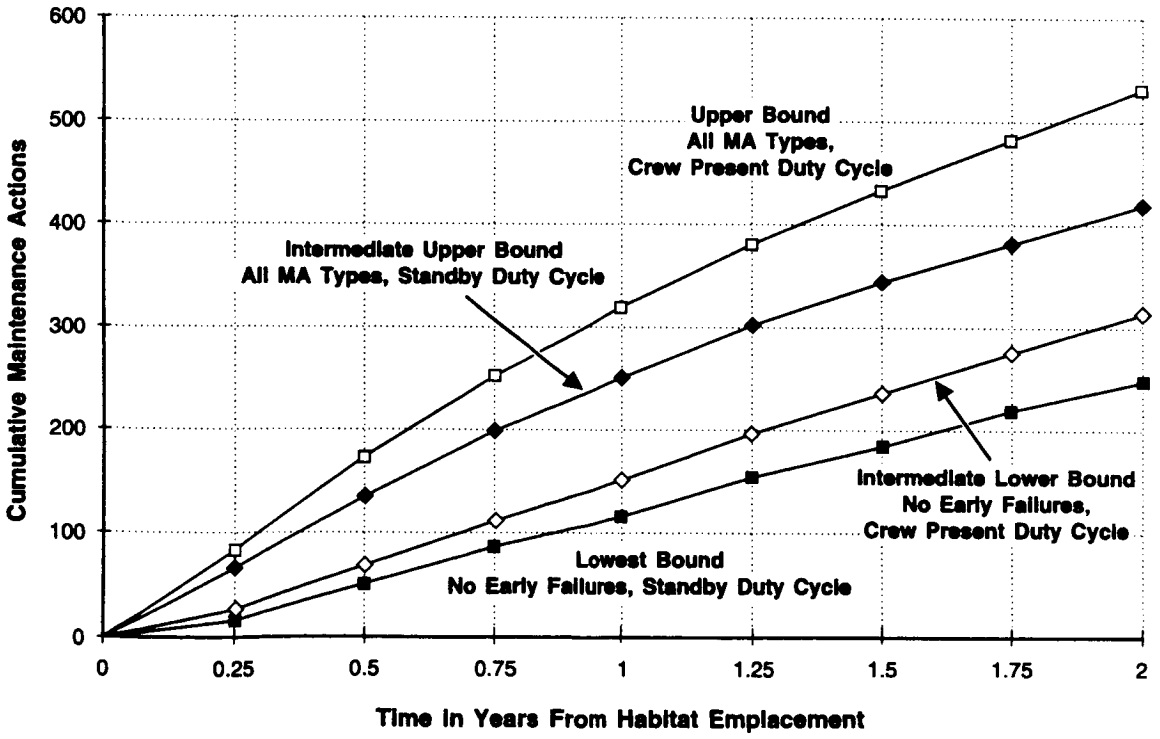


Figure 3 – Bounds for total maintenance actions – instantaneous replacement.

to the standby duty cycle plots. Separate EVA and IVA results are estimated but are not shown here.

Table 2 gives the mean number of maintenance actions for a variety of scenarios. The numbers are not cumulative. For example, the number of maintenance actions identified prior to the crew landing on the lunar surface is listed in the instantaneous replacement scenarios in the "visit 1 before" column. The number of new maintenance actions that arises while the crew is present is listed in the "visit 1 crew" column. The number of new maintenance actions that occurs after the visit 1 crew has departed the lunar surface until the time the second crew visits the lunar surface is listed in the "visit 2 before" column. For the scheduled resources scenarios, the "before" columns show the number of maintenance actions identified up to 2 weeks prior to the crew landing on the lunar surface. We speculate that the crew brings replacements for this set of maintenance actions. From these numbers, the backlogs can be calculated.

Table 3 gives the EVA and IVA requirements for maintenance by crew based on maintenance actions identified prior to crew departures from Earth. This scenario corresponds to a logistics support of carrying spares for failures diagnosed up to about 1 week prior to crew departure. Results are shown for 2-month and 6-month delays and for the first three visits to the outpost. Values given are for the maintenance actions identified before crew departure. The new maintenance actions that occur after crew departure from Earth through the time of return from the Moon are backlogged until the following crew visit. (In the 2-month scenarios, the number of EVA and IVA shifts backlogged to visit 2 exceeds the EVA's and IVA's that are in the "visit 1" column.) In the scenarios that include all failure types, the number of required EVA's exceeds the EVA allotment through all three visits. IVA shifts required are within the allotment for visit 1 but would exceed the same allotment for visits 2 and 3.

Table 2 – Number of maintenance actions.

Model scenario	Delay before first crew visit	Visit 1		Visit 2		Visit 3	
		Before	Crew	Before	Crew	Before	Crew
Instantaneous replacement, all MA types	2 months	42	56	127	36	99	31
	3 months	76	39	124	34	98	36
	4 months	96	40	118	43	89	31
	6 months	136	52	113	26	85	33
Scheduled resources, all MA types	2 months	31	63	93	41	87	29
	6 months	111	46	106	32	82	20
Scheduled resources, no early failures	2 months	12	33	41	16	61	16
	6 months	49	18	62	15	61	17

Before = Number prior to crew arrival, since last crew departure
Crew = Additional number occurring during crew visit

Table 3 – Number of EVA and IVA shifts required to perform maintenance actions identified prior to crew departure.

Model scenario	Delay before first crew visit	Visit 1		Visit 2		Visit 3	
		EVA	IVA	EVA	IVA	EVA	IVA
Scheduled resources, all MA types	2 months	5	3	12	14	10	12
	6 months	10	10	12	14	9	11
Scheduled resources, no early failures	2 months	3	1	6	7	6	7
	6 months	5	5	7	7	7	8

We have made a preliminary assessment of the functional impact each scenario has on the FLO habitation element. Although RMAT has the capability, our representative data base does not include functional block diagrams or comprehensive criticality identifiers. Redundant LRU's have shared duty cycles. The functional impact we have examined, as output by RMAT, is on the set of multiplexers/demultiplexers (MDM's). The MDM's are of particular significance because they provide the translation between the operative subsystems and the data management system for control and monitoring. Additionally, they are of sufficient number (31) to look at the results from a qualitative point of view. Scenarios with instantaneous replacement have no functional impact. Both the 2-month and 6-month delay scenarios with scheduled resources and all failure types have similar, and apparently significant, functional impacts. Approximately 15 percent of the time fewer than 50 percent of the MDM's are operating; 70 percent of the time fewer than 75 percent of the MDM's are operating. For the 2-month and 6-month scenarios with only random failures and preventive maintenance, 25 to 30 percent of the time fewer than 75 percent of the MDM's are operating.

The following are observations from the results of the simulations, including those discussed above. Unless otherwise specified, the observations are based on all MA types and for maintenance actions only by crew, with backlogs of maintenance actions not diagnosed prior to crew lander readiness (spares loaded 2 weeks before crew landing on the lunar surface).

- The number of maintenance actions for visit 1 is sizable, regardless of scenario, and ranges from 45 (2-month delay, scheduled resources, no early failures) to 188 (6-month delay, all MA types, instantaneous replacement). Furthermore, the number of maintenance actions for instantaneous replacement and for scheduled replacement is in the same ballpark; i.e., 98 versus 94 for first visit, 2-month delay and 445 versus 397 for three visits, 6-month delay (see Table 2).

- Except for the first visit of the 2-month delay scenario, the greatest demands for maintenance actions occur while crew is not present (see Table 2).

- The crews will be faced with a sizable backlog of (prediagnosed?) maintenance actions upon arrival and will have to contend with significant additional maintenance actions that occur after their departures from Earth (see Table 2).

- There are significantly fewer maintenance actions for the first visit if the time delay between habitat landing and crew landing is reduced (e.g., 94 for 2-month delay versus 157 for 6-month delay). But the number of these maintenance actions that occurs after the crew lander is ready for launch is greater for the reduced delay; e.g., 63 for 2-month delay versus 46 for 6-month delay (see Table 2).

- For delays up through 6 months, the peak number of maintenance actions occurs on the second visit (see Table 2).

- The number of IVA maintenance actions is greater than the EVA maintenance actions by a factor of 2 to 3 (interior LRU's outnumber exterior LRU's by approximately 7 to 1). However, the EVA total demand time (work site plus overhead plus sortie time) will be similar to the IVA total demand time for reasonable levels of overhead times (0.5 hr IVA, 1.0 hr EVA). Demand time is not shown here.

- The allocation of 10 shifts for IVA maintenance for the FLO first visit is sufficient to satisfy the demand, except for the maintenance actions arising after crew departure (6-month delay scenario). Additional allocation of IVA shifts will be required for visits 2 and 3 (see Table 3). (The allocation may be sufficient, depending upon further specifics of IVA definition.) All visits of scenarios without early failures fall within the allotment of 10 IVA shifts.

- A FLO first visit allotment of four EVA's for maintenance will not be sufficient; 5 to 10 EVA's will be required plus whatever is required to contend with the maintenance actions that will be backlogged. An even larger number of EVA's will be required on visits 2 and 3 because of the backlogged maintenance actions from previous visits (see Table 3).

3.1.2 Implications of Results

Implications derived from the simulations can provide early insight into the FLO mission design and the role of automation and robotics. Significant implications for FLO include the following:

- Science, exploration, and technology objectives will be impacted unless maintenance demands on the crew are minimized.

- The indicated number of maintenance actions will have a significant impact on logistics resupply and resources (spare parts, EVA's, IVA shifts, robots, data system, etc.).
- There will be significant impact to the functionality of the outpost if timely repairs are not made.
- The number of new maintenance actions after crew departure from Earth indicates special attention to levels of redundancy and commonality to the lowest level is indicated.
- Timely, reliable diagnosis of failures will be critical and must be designed for.
- The number of maintenance actions when the crew is not present must give rise to serious considerations for robotic repair capabilities.
- Design for ease of maintenance and repair will be important in minimizing crew (or robotics) maintenance demands.
- An onboard maintenance capability (workstation, tools, equipment, etc.) is indicated.

Several factors (sizable number of maintenance actions, majority when crew is not present, work site and overhead times, impacts on exploration and science) indicate a real need for robotics. Robotics are needed to

- Support diagnosis (test and inspection) for both EVA and IVA.
- Assist the crew by transporting and positioning parts, sensors, and tools and possibly positioning the crew.
- Perform robotic maintenance to minimize demands on crew, minimize backlogs of maintenance between crew visits (some maintenance will still require crew involvement), and free up the crew for science and exploration activities.
- Perform dusting, servicing, etc.

In addition to the maintenance actions described in this study, there will be other maintenance actions, including dusting of sensitive surfaces and repairs to parts not characterized as LRU's. These may be infrequent but time consuming. Maintenance of the rover, crew lander, and scientific instruments will also be required.

The results presented here were the first simulation results of the FLO mission and have demonstrated the merit of early simulations to evaluate mission feasibility. As the mission definition changes because of these results and other considerations, additional simulations should be made. The iterations of simulations with mission designs early in the mission definition stages can be of significant impact in making the mission feasible.

Requirements are characterized early when they can have the most benefit at least cost. FLO and all similar mission scenarios should adopt a design for reliability and maintainability early in the program. This design should include, as a minimum, consideration of provisions for the following:

- EVA and IVA repair robotics
- Full fault diagnosis, meaning design for diagnosis
- Critical levels of redundancy
- Commonality to the lowest level of design
- Provisions for spares and logistics
- Design for ease of maintenance and repair
- Adequate sensing and testing equipment
- Tools and equipment for maintenance
- Maintenance workstations
- Crew availability and training
- A knowledge support system

4. Advanced Life Support System Robotics

Neither of the above studies explicitly addressed the use of robotics to solve the problem of excessive crew time being required to operate various "subsystems," such as power, communications, thermal control, and life support for a permanently manned outpost. We briefly address life support here.

Since 1978 NASA has studied closed and controlled ecological life support systems (CELSS) or advanced life support systems (ALSS), which are bioregenerative and based on a combination of biological and physicochemical components that may be used on future missions in low-Earth orbit, in transit to other planetary bodies, and on lunar and planetary surfaces. Higher plants will be used in food production, water purification, carbon dioxide uptake, and oxygen release.

Agriculture can be very labor intensive or assisted by automation (robotics). Operations of an ALSS such as crop seeding, nutrient solution maintenance, transplanting, plant observation,

harvesting, edible biomass separation, transporting, and preventative maintenance, if carried out by intelligent robotic systems, could greatly reduce the excessive crew time requirements to a reasonable level. Ten crops are apparently needed to supply nutrition needs. JSC is working toward a year-long, high fidelity test in a Human Rated Test Facility (HRTF) with four 90-day stays by a crew of four aided by automation and robotics.

Experience from the Russian BIOS 3 experiments [19] indicated an average of greater than 4 hr per day for each crewmember was required to deal with food production. Biosphere 2 results indicated an average between 2 and 3 hr per day for each of eight crewmembers was required to operate the food production aspects. Intelligent robotics can be used to reduce these times to an acceptable minimum for the HRTF [20].

5. Usefulness of the Technology on Earth

Space robotics, especially those systems being developed to operate on planetary surfaces, can be considered a form of the emerging technology of field robotics on Earth. The solutions to the problems we will be solving to make the exploration of our solar system possible and practical will apply to the many problems we have that require operating in hazardous environments on Earth and critically improving human productivity in many fields. Service industries can also use these developments in relatively unstructured environments.

Compared to the applications of space robotics in the Shuttle or on Space Station, the supervised autonomous robotics needed to make space exploration planet surface activities reliable and productive are closer to the capabilities required on Earth for many productivity improvements that raise the standard of living for everyone. The greatest benefit to the U.S. economy of any space exploration related technology can come from the development of supervised autonomous systems [21].

6. Conclusions

Several conclusions are suggested by the results presented in this paper. One is that space exploration will be "enabled" by the use of supervised intelligent systems on the planet surfaces, including supervised autonomous robotic systems. With sufficient use of intelligent systems, the number and skills of humans on the planet surface will be determined predominantly by surface exploration, science, and local resource use (not outpost) objectives and requirements. Several other uses of intelligent systems in Earth orbit or during space transport are indicated by current studies.

Additional modeling studies should be carried out to provide further results and insight. Our MSAT modeling tool makes these studies easier to do relatively quickly.

Another conclusion is that more definitive requirements definition studies should be carried out for space exploration supervised intelligent (autonomous) robotic systems.

7. References

1. The Synthesis Group. "America at the Threshold." Superintendent of Documents, U.S. GPO, Washington, DC, May 3,1991.

2. Erickson, Jon D.; et al. "SEI Planet Surface Systems Humans/Automation/Robotics/ Telerobotics Integrated Summary." JSC-45100, NASA JSC, Houston, TX, Mar. 1991.

3. Erickson, Jon D.; et al. "A&RD Response to the Synthesis Group Report." Briefing to Level II Lunar and Mars Exploration Program, NASA JSC, Houston, TX, July 17, 1991.

4. Bureau of Export Administration. "National Security Assessment of the U.S. Robotics Industry." Department of Commerce, Washington, DC, Apr. 1991.

5. Engelberger, Joseph F. "Robotics in Service." MIT Press, Cambridge, MA, Jan. 1991.

6. Erickson, Jon D. "The Role of Intelligent Systems in Space." Proceedings of International Conference AI91: Frontiers in Innovative Computing for the Nuclear Industry, American Nuclear Society, Jackson, WY, September 1991.

7. Erickson, Jon D.; et al. "An Intelligent Space Robot for Crew Help and Crew and Equipment Retrieval." International Journal of Applied Intelligence, accepted for publication in 1994.

8. IEEE Transactions on Systems, Man, and Cybernetics. "Special Issue on Unmanned Vehicle and Intelligent Robotic Systems." Vol 20, No. 6, Nov./Dec. 1990.

9. Aucoin, P. J.; and Graves, P. L. "SEI PSS H/A/R/T Option 5A Mission Analysis." JSC-45105, NASA JSC, Houston, TX, Mar. 1991.

10. Cohen, Aaron. "Report on the 90-Day Study on Human Exploration of the Moon and Mars." NASA JSC, Nov. 1989.

11. Erickson, J.D.; Aucoin, P.J., Jr.; and Dragg, J.L. "A Tool for Simulating SEI Operations As a Means of Obtaining Intelligent System Requirements." Proceedings of SPIE 1829 Conference on Cooperative Intelligent Robotics in Space III, Boston, MA, Nov. 1992.

12. Ellery Systems, Inc. "C-Lite Function Reference Guide for Ellery Open Systems." Boulder, CO, Feb. 1992.

13. Fisher, William F.; and Price, Charles R. "SSF External Maintenance Task Team Final Report." NASA JSC, Houston, TX, July 1990.

14. Katzberg, Steven J.; et al. "Aerobrake Assembly With Minimum Space Station Accommodation." NASA Tech Memo 102778, NASA Langley Research Center, Hampton, VA, Apr. 1991.

15. Erickson, J.D.; et al. "Some Results of System Effectiveness Simulations for the First Lunar Outpost." AIAA 93-4136, AIAA Space Programs and Technologies Conference and Exhibit, Sep. 1993.

16. Livingston, W.L. "TQM, Total Quality Management." Conference of the Quality Assurance Institute, Washington, DC, May 22, 1991.

17. Blumentritt, W. "The Space Station Freedom Reliability and Maintainability Assessment Tool and Its Use for Prediction of Maintenance Demand." NASA JSC Reliability and Maintainability Division Analysis Activity, May through July, 1992.

18. Space Station Freedom Program. "Assembly and Maintenance Implementation Definition Document (AMIDD)." NASA, Sep. 15, 1991.

19. Schwartzkopf, Steven H. "Prioritizing Automation and Robotics Applications in Life Support System Design." Lockheed Missiles and Space Co., Sunnyvale, CA, 1992.

20. Miles, G.E.; and Erickson, J.D. "Robotics in Controlled Environment Agriculture." Proceedings of AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space, Houston, TX, Mar. 1994.

21. Office of Technology Assessment. "Exploring the Moon and Mars: Choices for the Nation." Congress of the United States, Washington, DC, Aug. 1991.

A MULTITASKING BEHAVIORAL CONTROL SYSTEM FOR THE ROBOTIC ALL TERRAIN LUNAR EXPLORATION ROVER (RATLER)

P. Klarer*
Sandia National Laboratories
Albuquerque, New Mexico

Abstract

The design of a multitasking behavioral control system for the Robotic All Terrain Lunar Exploration Rover (RATLER) is described. The control system design attempts to ameliorate some of the problems noted by some researchers when implementing subsumption or behavioral control systems, particularly with regard to multiple processor systems and real-time operations. The architecture is designed to allow both synchronous and asynchronous operations between various behavior modules by taking advantage of intertask communications channels, and by implementing each behavior module and each interconnection node as a stand-alone task. The potential advantages of this approach over those previously described in the field are discussed. An implementation of the architecture is planned for a prototype Robotic All Terrain Lunar Exploration Rover (RATLER) currently under development, and is briefly described.

Introduction

One of the more interesting problems in the fields of robotics and artificial intelligence research is the autonomous navigation and control of ground vehicles. The capability for a vehicle to autonomously perceive, plan, and navigate through obstacle fields in realistic conditions has potential application in a wide variety of areas, from automated warehouse delivery carts to planetary exploration platforms. Since the advent of modern digital computers the typical approach to solving the autonomous navigation problem has been through Artificial Intelligence (AI) techniques, where a systematic procedure of perception, modeling, planning, action and feedback (see Figure 1) are applied in a manner reminiscent of human being's problem solving techniques[1,2].

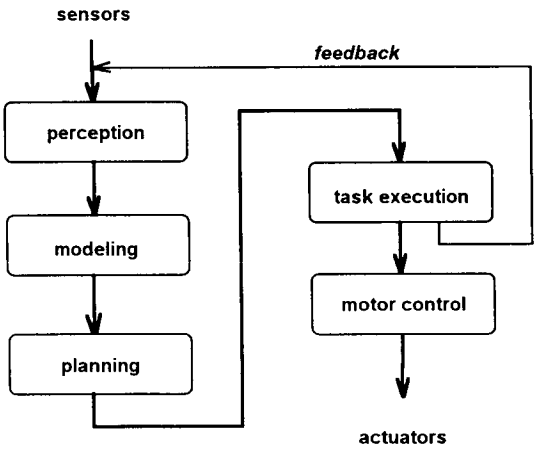


Figure 1. Traditional AI Approach
(from Brooks, 1986)

The computational requirements for such systems have proven to be significant, and alternatives have been developed over the past few years which employ techniques that are less anthropomorphic, including neural networks, fuzzy logic, and behavioral control. Behavioral control was first proposed by Brooks[3] as a simple and extensible architecture that could provide fully autonomous systems without the high

* Member of the Technical Staff, Advanced Vehicle Development Department, Robotic Vehicle Range

This work is sponsored by the Laboratory Directed Research and Development (LDRD) Program at Sandia National Laboratories and the US Department of Energy.

Copyright c 1993 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

computational requirements of the traditional AI approach. The subsumption approach is based on an entirely different paradigm which is less 'sequential' in nature than traditional AI, and is more 'layered' (see Figure 2).

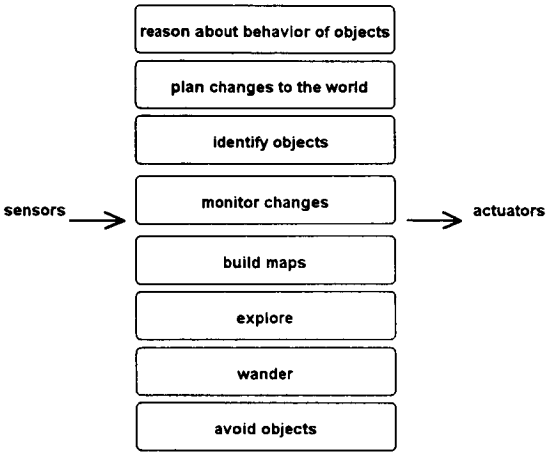


Figure 2. Subsumption Approach
(from Brooks, 1986)

Some observations by researchers who have developed these behavioral or 'subsumption' based systems have pointed out pitfalls in the subsumption paradigm that can make implementation problematic[4]. The use of multiple processors that are physically wired together to produce the subsumption system resulted in a system that was mechanically complex, required custom designed hardware, and was somewhat prone to failure. Although these problems may be alleviated through the miniaturization and consolidation of the 'macro-sized' multiprocessor system to a 'micro' or even 'nano' sized custom silicon chip[5], the basic weakness of these systems is their inability to provide internal models or representations of the environment. This lack of 'state' information makes interaction with the system by a user somewhat like attempting to converse with a very simple lifeform. It must be stated that the elimination of internal models or 'state' information was one of the objectives of the subsumption architecture's proponents, in that it is precisely this requirement for accurate world models that drives the traditional AI techniques' high computational requirements, and the argument has been made that an insect requires very little 'computation' power to function autonomously[6]. While it is true that insects can function autonomously in a real world environment, it is difficult to imagine a technique that allows one to interactively direct the behavior of an insect to suit one's purpose, and that is after all, one of the important criteria for robotic systems in the service of humankind. One potential approach that not only circumvents those problems but may actually facilitate the implementation of a subsumption architecture is to employ a multitasking framework as the basis for system design. This approach employs conventional computer processors to alleviate the mechanical complexity issue, allows software modules to operate independently in real time much as the subsumption architecture's 'behaviors' do, and provides many intrinsic features that facilitate the communication and interactions between independent modules. In addition, a real-time multitasking system allows the use of traditional AI problem solving techniques in concert with a subsumption architecture, resulting in a hybrid system that takes advantage of the benefits of both approaches. The remainder of this paper is devoted to describing how such a system could be implemented, and discusses some concepts for specific implementation on a mobile robotic system under development at Sandia National Laboratories' Robotic Vehicle Range. It should be noted that detailed explanations of the subsumption approach are referenced at the end of this paper, and are therefore not covered here, however Dr. Brooks' notation is used herein to describe the use of the subsumption architecture's connections and features.

Generalized Hybrid System Description

Figure 3 illustrates a generalized hybrid system architecture for robotic system control, employing both a subsumption system (surrounded by a dashed line), and a conventional set of real-time input\output tasks and associated data stores. Data acquisition from the sensor suite is handled by one or more specialized tasks, which typically run at a fixed rate. The fixed, regular rate of data sampling is an important characteristic of real-time systems, in that the application of some digital filtering or

post processing techniques often requires that data from several disparate sensors be acquired at fixed, repeatable intervals. One of the major advantages of a real-time multitasking system in this context is the ability to add, modify or delete major modules without affecting the timing characteristics of the system as a whole, or the iteration rate of other task modules in the system. As shown in the figure, data acquisition is completely independent of the subsequent tasks in the system that make use of it, as are the control output tasks.

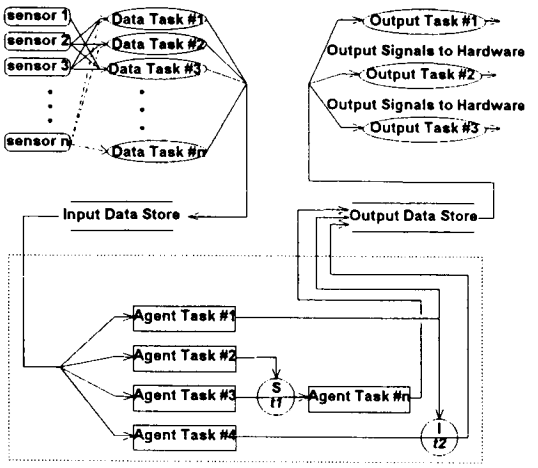


Figure 3. Multitasking Behavioral System Architecture

By using a set of common data stores between the I/O tasks and the subsumption control system, the architecture allows for the addition of other 'high level' control or 'AI-like' elements, as shown in Figure 4. For example, interfacing a set of tasks which perform perception and feature extraction, the maintenance of a data-based world model, and the planning of actions based on the model to the system is straightforward, and only affects the system at two points. The input data store must accommodate any new sensors required for perception, and the motion planner's output is hooked into one of the subsumption system's outputs via a new subsumption node. The world model data store is entirely independent of any other features of the system, and is only interfaced to the new tasks which require it.

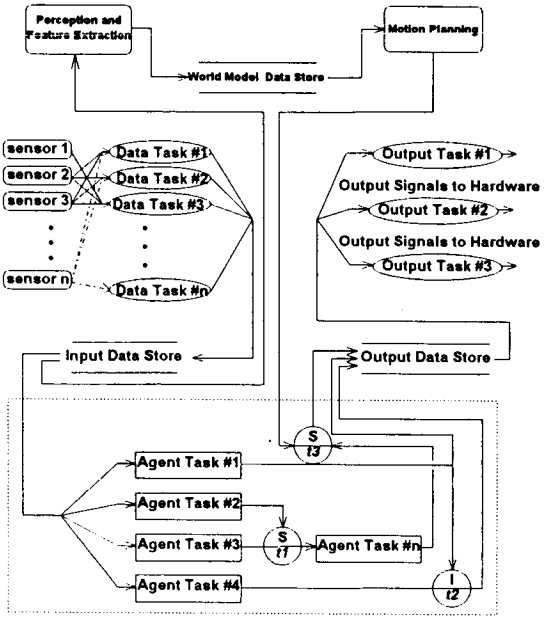


Figure 4. Hybrid System Architecture

The previous example is included only to illustrate the method by which a subsumption system can be integrated with traditional AI approaches, and how a real-time multitasking architecture facilitates this. In an actual implementation, the addition of traditional AI techniques may be obviated if the subsumption system is fully implemented, resulting in a fully autonomous system which is capable of initiating motion, avoiding obstacles, pursuing some goal directed behavior, perception and\or reasoning about disparate sensory input, and exploration of its surrounding environment.

As noted in Brooks' description of the subsumption architecture, communications between each of the modules and the method of interconnecting the various modules through special nodes is the key to implementation[3]. The built-in features of most multitasking systems that allow intertask communication and synchronization provide the methods for implementing a subsumption system such that more than a single specific mechanism may be used. The most common feature available in multitasking systems is message passing. Message passing involves packaging data into a special data structure specific to the operating system, and tagging the data structure's 'destination'

variable (or pointer) so that the multitasking kernel knows which task is the intended recipient. The kernel is invoked with a special call by the sender, and the message arrives at the destination task at some time in the future. In a real-time multitasking system, the message 'delivery' time is guaranteed to be no more than some specific interval. For the purposes of a subsumption system, the contents of the message would be the pertinent data from one module's 'output line' as described by Brooks, which corresponds to an 'input' line to either another module or to a subsumption node. In either case, the embodiment of behavioral modules and nodes as separate tasks makes them appear to be identical to the multitasking kernel, and are treated in exactly the same way by the system. Thus, message passing provides an immediate interconnection tool to create a subsumption system using tasks to implement behavior modules and connection nodes, and potentially allows both synchronous and asynchronous functionality, depending on the details of the multitasking kernel's features.

An additional, complementary method for performing intertask communications involves the use of a user-defined data structure located in global memory (or accessed via a pointer) and a real-time multitasking feature called an 'event'. Events are implemented in a variety of ways across different kernels, but usually involve a special bitfield or kernel variable that is monitored by the kernel for changes. Changes in the state of the event bitfield are initiated by tasks that wish to record the fact that something of importance has occurred, and some other task needs to respond. The kernel notes a change in the event bitfield at some defined, minimum time after a task changed the state of the bit, and responds by triggering or 'kicking' the second task which is hooked to the event bit. Although somewhat clumsy when compared to message passing, the use of real-time 'events' and an associated data structure operates much faster than the passing of a message. This is due to the fact that message passing usually involves much more in the way of data operations by the kernel than the simple setting or clearing of a single event. Either way, the intrinsic features of a real-time multitasking system provide the means by which synchronous and asynchronous interactions between independent tasks may be applied to implementing a subsumption architecture.

Example Implementation for RATLER

In the previous section some generalized descriptions of multitasking features were used to illustrate how a subsumption system could be implemented within such a framework. In this section, an example implementation for a specific mobile robot system is described. The example is taken from Brooks' description of a subsumption system[3], and has been modified to fit the multitasking approach to implementation for the Robotic All Terrain Lunar Exploration Rover (RATLER)[7]. The RATLER II vehicle shown in Figure 5 is a four wheeled, skid steered articulated chassis design, with the body divided into right and left halves, two wheels on each side. The halves are joined together such that they may rotate along the lateral axis to enhance the mobility and stability of the platform. The RATLER is intended for use as a teleoperated vehicle, with semiautonomous navigation capabilities selectable by a remote operator. The RATLER will eventually carry a suite of instruments to perform mission-specific tasks, and will also be fitted with a multi-DOF manipulator arm. The primary mission of the RATLER is planetary exploration, but the platform may find terrestrial applications as well.

Figure 5. RATLER II Prototype

Although not yet implemented, some form of proximity sensing will be required on the vehicle to provide obstacle detection input for the software control system. This will most likely take the form of a series of infrared or microwave range sensors arrayed around the vehicle's periphery, which output a signal if an object is detected within a preset minimum range limit or set of range bins.

The subsumption architecture as originally described by Brooks does not include any provisions for a user to interact with the system directly. Rather, the focus on minimalism to achieve autonomy ignores the possibility of teleoperation as a control mode. For certain applications, i.e. the exploration of a planetary surface too far removed from a human teleoperator for real-time remote control to be feasible, a strictly autonomous operations mode makes sense. However, there are many other applications of robotic vehicles that would benefit from a capability to teleoperate, including both terrestrial and possibly lunar surface missions[8,9]. Two implementation concepts for a subsumption system with a teleoperation capability are briefly described below. The first assumes teleoperation to be the 'lowest' functional layer in the subsumption architecture, whereas the second concept assumes teleoperation to operate at the 'highest' level. Precisely which of the two approaches, or some other as yet undefined approach, is best will the subject of future work.

Teleoperation as the 'lowest' level:

As illustrated in Figure 6, the subsumption system proposed for the RATLER is based on the real-time multitasking approach described previously. A set of dedicated tasks perform data acquisition from a sensor suite, and run at a fixed, periodic rate between 10 and 100 Hz. The timing interval is controlled by the multitasking kernel via inherent timer interrupts, and is transparent to the user. The sensor data is stored in a common memory structure and is available to the subsumption system's modules as required. Each module accesses only the data it needs in order to perform its function, which is to operate as a finite state machine. The data store is accessed via pointers, and does not require any event bits or message passing. Note that this system differs from Brooks' original description in one significant regard; that is, the lowest functional element is a module called 'teleoperate'. This is assumed to be the lowest possible level of functionality, considering a teleoperation system to be of a lower order than a self guided or autonomous system.

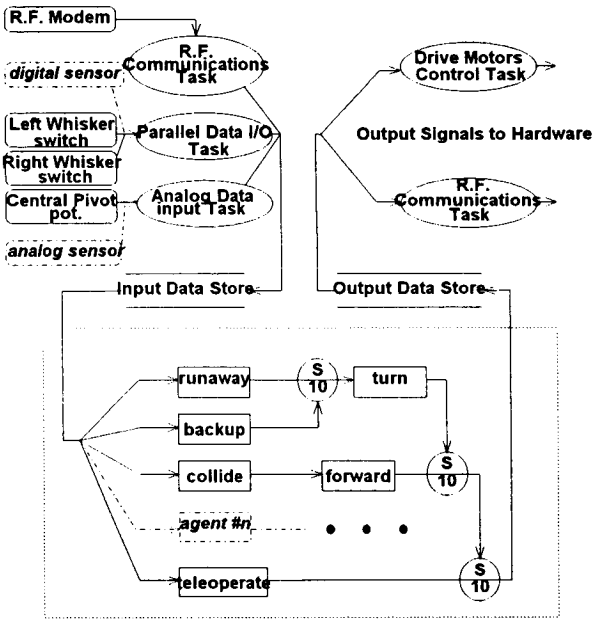


Figure 6. Teleoperation as 'lowest' level

As the lowest level of functionality in the subsumption system, teleoperation is always active if commands from the user (located at a remote control console) are arriving and are being stored in the data structure. If any of the 'higher' modules illustrated in Figure 6 are active, their outputs will subsume the teleoperator's outputs in a manner identical to that described for Brooks' subsumption architecture. This simple change from the original system description allows a direct user interface for teleoperation, and does not adversely affect the operation of the overall subsumption design. The higher subsumption modules may be switched 'on' or 'off' by the user via a set of flags in the data structure, so that the system can be operated in either a straight teleoperation mode, or in any combination of autonomous modes, depending on which behaviors are switched on and which are switched off.

Each behavior module is an independent task which runs at a specific periodic rate, typically between 10 and 100 Hz, and communicates with other modules via either the event bit method or message passing method as previously described. The subsumption nodes are designated as circles with an 'S' in the figure, and are also implemented as independent tasks which run at fixed periodic rates. In order to perform their function as subsumption nodes, they must maintain an internal record of the latest inputs from their associated behaviors, and perform a finite state machine transition based on their simple set of internal rules. The result of the machine state switch determines what the node's output signal will be, and that is sent via either a message or event bit signal to the next behavior or node as illustrated above. After the signals have passed through the subsumption system and have arrived at the output data store, placed there via pointer access, a set of dedicated output control tasks access the data structure for setpoints and translate them into output signals to the robot's hardware. Functionally, this system is no different than more conventional implementations of the subsumption architecture, except it relies on the inherent facilities of a real-time multitasking kernel to perform the communications required between modules and nodes, and to control the real-time preemptive triggering of those modules according to a real-time clock.

Teleoperation as the 'highest' level:

In contrast to the concept previously described for a teleoperation capability in a subsumption system, the assumption of teleoperation as occupying the 'highest' level in the subsumption system leads to a very different implementation scheme. The system shown in Figure 7 is the basic multitasking subsumption architecture described in earlier sections of this paper, with the addition of a teleoperation module *outside the subsumption subsystem*. The teleoperation module receives its inputs from a remote control station where a human operator is located, whose commands are in the form of motor\axis setpoints in position, velocity, or torque. Global commands in the form of a desired vehicle heading or destination waypoint may also be generated from the remote console.

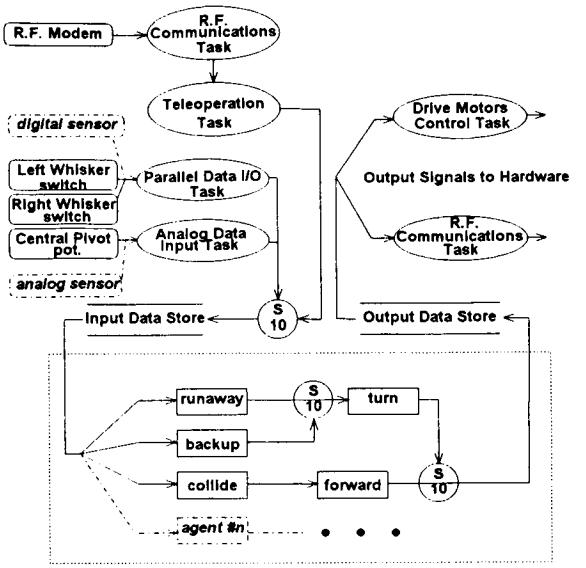


Figure 7. Teleoperation as 'highest' level

The desired setpoints or global state commands received by the teleoperation task are used to generate a set of synthetic alternate signal inputs from appropriate sensors, which are then placed in the input data store. They may simply be used to modify real sensor data, or to replace the real data altogether much in the same fashion as the subsumption subsystem operates on its data flows. The act of teleoperating then becomes a matter of 'faking out' the subsumption system by providing it input signals that will cause the subsumption system to react in the desired manner. A simple example would be to cause the vehicle to move forward, the generation of a synthetic obstacle signal from one of the rearward facing obstacle sensors by the teleoperate task should cause the 'runaway' behavior to activate, and the subsumption system would then move the vehicle away from the offending signal, in other words, forward. This approach may have advantages over the previously described 'approach, in that it relies on the subsumption system to perform all control outputs and is implemented outside of the subsumption subsystem, whereas the 'teleoperation is lowest' method requires the teleoperation task to be an

integrated part of the subsumption architecture.

Summary

This paper has described an approach to implementing the so-called 'subsumption' or 'behavioral' control scheme for robotic systems within the framework of a real-time multitasking architecture. The potential advantages of this system result from taking advantage of timing and communication features available with most multitasking kernels. The proposed architecture also allows for the construction of hybrid systems which employ both subsumption and traditional AI techniques, and easily provides for a teleoperator's interface as well. The proposed system is well suited to operating on conventional general purpose computing hardware, and should allow development with existing software tools designed for real-time multitasking systems.

Future Work

Although only two multitasking features were described, event scheduling and message passing, there are undoubtedly other inherent multitasking system features that may be employed to some advantage. The benefits of employing teleoperation as either the 'highest', 'lowest' or some intermediate level with respect to the subsumption system remains to be fully evaluated. Pursuing those and other issues will be part of future efforts by Sandia National Laboratories in the development of the RATLER control system software.

Acknowledgements

Special thanks go to P. Heerman, R. Byrne, B. Pletta, and W. Amai for their assistance in developing the ideas in this paper. Their insights and unconventional approaches to problem solving will hopefully help bring these ideas to fruition in the future, as they already have helped to bring the basic concepts into communicable form. The author also acknowledges the work of the people of the Artificial Intelligence Laboratory at MIT, without whose groundbreaking efforts this paper would not have been inspired.

References

1. CROWLEY, J. L., "Navigation for an intelligent mobile robot.", IEEE Journal of Robotics & Automation, vol. RA-1, no. 1, March 1985.

2. MORAVEC, H. P., "The Stanford cart and the CMU rover.", Proceedings, IEEE, vol 71, July 1983.

3. BROOKS, R. A., "A Robust Layered Control System for a Mobile Robot.", IEEE Journal of Robotics & Automation, vol. RA-2, April 1986.

4. CONNELL, J. H., "Minimalist Mobile Robotics, A Colony-style Architecture for an Artificial Creature", Academic Press Inc., San Diego, CA, 1990.

5. BROOKS, R. A., "Micro-Brains for Micro-Brawn; Autonomous Microbots", Proceedings, IEEE Micro Robots and Teleoperators Workshop, November 1987

6. BROOKS, R. A., "Engineering Approach to Building Complete, Intelligent Beings", SPIE Vol. 1002 on Intelligent Robots and Computer Vision, 1988

7. PURVIS, J., and KLARER, P, "R.A.T.L.E.R.: Robotic All Terrain Lunar Exploration Rover," Sixth Annual Space Operations, Applications, and Research Symposium, Johnson Space Center, Houston, TX, 1992.

8. MEYER, C., "Goals and Requirements for Scientific Lunar Rovers", Proceedings, ASCE, SPACE 94: Conference on Robotics for Challenging Environments, February 1994.

9. HOFFMAN, S.J. and WEAVER, D.B., "Results and Proceedings of the Lunar Rover/Mobility Systems Workshop", Exploration Programs Office document #EXPO-T2-920003-EXPO, NASA Johnson Space Center, Houston Tx, 1992

Thursday
March 24, 1994

SAVA III:
A Testbed for Integration and Control of Visual Processes

James L. Crowley
LIFIA - IMAG
46 Ave Félix Viallet
38031 Grenoble, France

Henrik Christensen
Laboratory of Image Analysis, Aalborg University
Fr. Bajers Vej 7 D
DK-9220 Aalborg, Denmark

1 Introduction

During the last few years, there has been a growing interest in the use of active control of image formation to simplify and accelerate scene understanding. Basic ideas which were suggested by [Bajcsy 88] and [Aloimonos et al. 88] has been extended by several groups including [Ballard 91], and [Eklundh 92]. This trend has grown from several observations. For example, Aloimonos and others observed that vision cannot be performed in isolation. Vision should serve a purpose, and in particular should permit an agent to perceive its environment. This leads to a view of a vision system which operates continuously and which must furnish results within a fixed delay. Rather than obtain a maximum of information from any one image, the camera is an active sensor giving signals which provide only limited information about the scene.

Bajcsy observed that many traditional vision problems, such as stereo matching, could be solved with low complexity algorithms by using controlled sensor motion. Examples of such processes were presented by Krotkov [Krotkov 90]. Ballard and Brown [Brown 90] demonstrated this principle for the case of stereo matching by restricting matching to a short range of disparities close to zero, and then varying the camera vergence angles. The development of robotic camera heads has lead to the possibility of exploiting controlled sensor motion and control of processing to construct continuously operating real time vision systems.

At the same time, research in applying artificial intelligence techniques to machine vision led to an emphasis on the use of declarative knowledge to control the perceptual process. Systems such as the Schema System [Draper et. al. 89] developed a black-board architecture in which multiple independent knowledge sources attempted to segment and interpret an image. A major problem in such systems is control of perception. Such systems emphasise explicit representation of goals and goal directed processing which direct the focus of attention to accomplish system tasks. It has not been obvious how such a knowledge based approach to control of attention could be married to a real time continuously operating system.

In July 1989, the European Commission funded a consortium of six laboratories to investigate control of perception in a continuously operating vision system[1]. The consortium partners set out to build a test-bed vision system for experiments in control and integration. An experimental test-bed system was constructed which integrates a 12 axis robotic stereo camera head mounted on a mobile robot, dedicated computer boards for real-time image acquisition and processing, and a distributed system for image description. The distributed system includes independent modules for 2-D tracking and description, 3-D reconstruction, object recognition, and control. On March 18 1992, a fully integrated continuously operating vision system was demonstrated to the European Commission using this test-bed. This paper reports on the development of this system and the research which the system makes possible in control of a real-time vision system. A more complete description of the results of the project may be found in the book [Crowley–Christensen 93].

1.1 The Project Vision as Process

The starting point for the project "Vision as Process" was the demonstration of an integrated vision system capable of continuous real-time operation. It was quickly realised that such an ambition raises two problems:

1) The technical problem of integrating processes which model the environment in terms of descriptions which are qualitatively different.

2) The problem of controlling the "attention" and processing of a continuously operating system.

Concerning the first problem, different robotic tasks require different kinds of descriptions of a scene. Such descriptions can include 2D image description, 3D scene descriptions and symbolic labelling of the components of a scene. Such processes are complementary and mutually supportive. A framework is required which would permit the integration of multiple vision processes. This can be considered an "engineering" problem.

[1]The Partners in project ESPRIT "VIsion as Process" are Aalborg University (DK), University of Surrey (UK), KTH, The Royal Institute of Technology (S), University of Linkoping (S), University of Genoa (I), LTIRF-INPG (F) and LIFIA-INPG (F).

Copyright© 1993 by James L. Crowley and Henrik I. Christensen. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

The second problem is both subtle and fundamental. Most of the algorithms used in vision have computational costs which depend on the quantity of data. In the best cases the relation is linear, but in many cases it is quadratic, cubic, or even exponential. Real time response requires that the processing time for any part of the system is limited. This requires that the amount of data considered during each processing cycle be bounded which raises the problem of which subset of the available data the system should attend during each cycle. This is part of the larger problem of controlling perception. General purpose real time vision system requires a solution to this problem.

These observations let the consortium to develop a long term work plan with both an engineering and a scientific goal. These are:

Engineering Goal:Develop techniques for integrating cyclic real time processes for description of a scene in terms of 2D images, 3D structure and labelled objects using active control of a camera head.

Scientific Goal: Develop methods (and a theory) of control of attention in perceptual processes.

With these twin goals, the consortium has developed a long term plan leading to the demonstration of methods for the construction of integrated continuously operating vision systems, and the elaboration of a theory for the control of such systems.

1.2 System Integration and Control

When the VAP project was conceived in 1988, only a small number of vision systems were capable of performing symbolic interpretation, and they were designed for interpretation of single (static) images. The well known examples included VISIONS [Hanson-Riseman 78], ACRONYM [Brooks 81], and 3DPO [Bolles-Horaud 84].

Most work on analysis of image sequences had been carried out on pre-recorded images and the level of description was almost entirely parametric. i.e., systems could describe regions or features with independent motion in terms of their image or 3D-velocity. A review of the state of the art is provided by Huang [Huang 83]. Continuous and real-time observation of a dynamically changing scene involves more than motion interpretation. A continuously operating vision process must be able to limit processing to a small subset of the data available from visual sensors, and to adapt its processing mode dynamically in response to events in the scene and requirements of the task.

1.3 The VAP Hypotheses

From its earliest meetings, the VAP consortium agreed that vision should be studied in the context of its purpose, i.e. its use by other processes. Without dedicating to any specific application, this implies that visual processing can be controlled to concentrate on the subset of visual information which is considered relevant to the current goal as defined by a user process. In addition, the consortium recognised the ability to exploit coherence in the dynamic evolution in a scene. In a continuously operating system, temporal context permits changes in the scene to be predicted and computational resources to be directed to confirm expectations. This implies that tracking is basic operation within a continuously operating system.

The goal of the VAP project is to demonstrate that a vision system must be designed as a continuously operating "process". To demonstrate this principle, the consortium has designed a six-year research program to develop techniques to interpret a dynamically changing, quasi-structured environment. These techniques exploit goal directed focus of attention involving controlled sensor motion and control of processing. Processing is directed by goals which change dynamically in reaction to the needs of the perceptual tasks and to events in the scene.

The following section reviews some previous approaches to integration and control. This previous work establishes the set of concepts and "prior art" from which the design of the VAP skeleton system draws.

2 Systems Architectures for Integration and Control

A suitable system architecture is required for experiments in integration and control of a continuously operating system. In this chapter, we review previous approaches to architectures of vision systems. From this review, we argue that flexible integration may be achieved through use of a standard module architecture, replicated at each of the levels in the system. Such a standard module architecture is described in more detail in chapter 3.

2.1 The Reconstruction Approach

A popular approach for structuring a vision system has been proposed by Marr [Marr 82]. Marr argues for a system defined around a hierarchy of representations: images, The primal sketch, the 2.5-D sketch (viewer centred depth map), 3-D map and symbolic description. In this model, processing is organised as sequential processes in which information flows up through the levels. Processing is data-driven, in the sense that recognition and description are based on descriptions constructed at the lower levels in the system. This model is computationally demanding and it has proven difficult (if not impossible) to provide image descriptors that are sufficiently robust to allow characterisation of all phenomena in a natural environment.

The Marr processing model may be termed a *reconstruction approach* as it aims at a full reconstruction of the environment. The processing model is purely data driven, and thus poses a problem in terms of computational resources. The Marr model

assumes all processing may be carried out as a sequential process. This implies that a module uses the representation(s) at the level just below as a basis for its processing and the result is stored in the next higher representational level. The interfaces are consequently well defined. A simplified model of the processing is shown in figure 1.

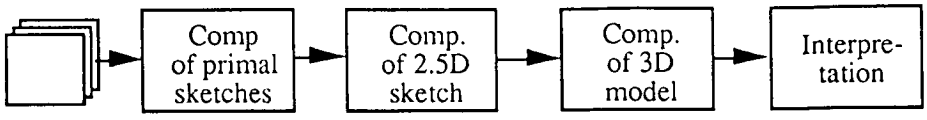


Fig. 1 A processing model for the reconstruction approach.

In terms of representations, this processing model implies that information needed to perform recognition and interpretation of settings must be available as part of the 3D model. i.e., a diverse set of descriptors must be tagged onto the 3-D model representation to facilitate recognition and description. This duplication of information up through the system and the unavailability of pixel level primitives can pose a problem in terms of model size and maintenance over time.

2.2 The Non-Committal Approach

In the VISIONS system [Hanson-Riseman 1978] data are stored on a blackboard, or common storage area and processing is performed in parallel by a number of "knowledge sources". All modules in the system can access information at any of the representational levels. This implies that information does not have to be replicated up through the system to be made available for recognition procedures. A simplified model of the non-committal approach is illustrated in figure 2.

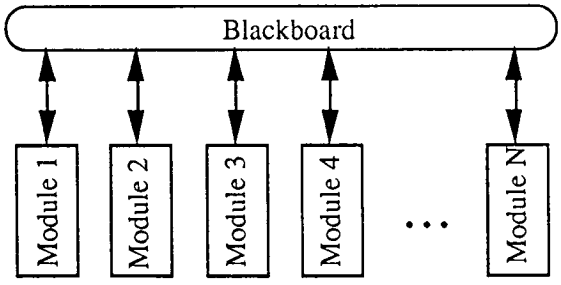


Fig. 2. Architecture for the non-committal approach.

In this architecture, each of the modules in the system has control information that specify the information that must be available before the module may carry out its task. In addition, it has control information that specifies the information which may be provided by the module. Through use of a control executive, it is possible to perform both goal directed and data driven processing through use of this control information.

This processing model imposes few constraints on the representations used in the systems and it is simple to add new modules to the system. The common blackboard is a potential problem for a continuously operating system. All information generated and used by the system passes through the blackboard. It might thus pose a problem with respect to information bandwidth. To indicate the amount of information, which may be generated directly from images one may look at the Image Understanding Architecture, described by Weems et al. [Weems 1989]. In that system, the storage reserved for intermediate representations is 4 GB and the system is only aimed at analysis of single images. Such an architecture will require extensive use of special purpose hardware, in particular when applied in the temporal context.

2.3 The Purposive Approach

Introduction of goal directed operation and use in a limited and well defined domain of application allow synthesis of a vision system which is composed of a set of specifically engineered modules. Such modules may be designed to be computationally well behaved, in the sense that the computational complexity is bounded and often robust representations can be provided. This approach to construction of vision system has been promoted by Bajcsy [Bajcsy 88], Ballard [Ballard 91] and [Aloimonos et al. 88]. Although the approach exploits specific vision modules, Bajcsy has tried to enumerate a set of modules that might form a general purpose system. The use of dedicated modules is a way to provide robust information and computationally tractable techniques. Well known examples of dedicated modules used for robot navigation are optical flow modules that can compute the position of the focus of expansion for the optical flow field, and modules which can compute the time to contact from motion in an image sequence.

This approach to system construction is termed *the purposive* or *the animate* approach. It is envisaged that the construction and analysis of specific modules will gradually provide insight that will allow definition of modules applicable in general vision systems. The convergence towards a standard set of modules through analysis of diverse application domains might provide valuable insight, but it is not obvious that convergence will be achievable.

In the purposive approach, the exploitation of information is task driven and may be very different from one task to the next. The basic system architecture should thus be flexible and facilitate dynamic change of the information flow. In practical systems a number of modules may exploit the same representation and once a system has been defined an

analysis of the representational requirements may point to the definition of a set of standard representations. Given present state of the art no such general representations are known.

A purely purposive approach to vision rejects most of the established techniques. Modules are to be built from scratch whenever a new type of information/representation is required. There is consequently a concern that from this approach little insight will be gained in terms the general vision problem.

2.4 The Vision as Process Architecture

During the first year, the consortium "Vision as Process" addressed the problem of design of an architecture which would meet the following criteria:

1) Continuously operating.

2) Integrating software contributions from geographically dispersed laboratories

3) Integrating description of the environment with 2D measurements, 3D models and recognition of objects.

4) Capable of supporting diverse experiments in gaze control, visual servoing, navigation and object surveillance.

5) Dynamically reconfigurable as the task changes.

The result was the design of a distributed test-bed system composed of independent modules. Modules may communicate by message passing over a central message server, or by dedicated "high-band width" channels. Systems can be composed from sub-sets of the available modules for individual experiments.

The architecture adopted by the consortium is shown in figure 3. The system has a data flow part, which is similar to the Marr processing model. It should be noted that the data flow is not only bottom-up but may also be top-down. Top-down expectations (derived from the present set of goals and contextual information) can be used to direct/control processing at lower levels, while detected event at the same time can drive a reconstructive mode of processing. The VAP architecture contains a common communication channel that allow communication between any pair of modules in the system. This communication channel may be used both for investigation of a non-committal processing model, and for investigation of purposive systems, as the component processes in the system in figure 3 may either general purpose or dedicated.

This architecture imposes few constraints on the design of component modules and it provides flexibility for the investigation of system level control issues. Initially it was envisaged that the main flow of data would exploit the communication links between adjacent modules, while only control information would be communicated through the common channel. During the execution of the project it was realised that a more flexible processing model was needed to make computations both efficient and robust.

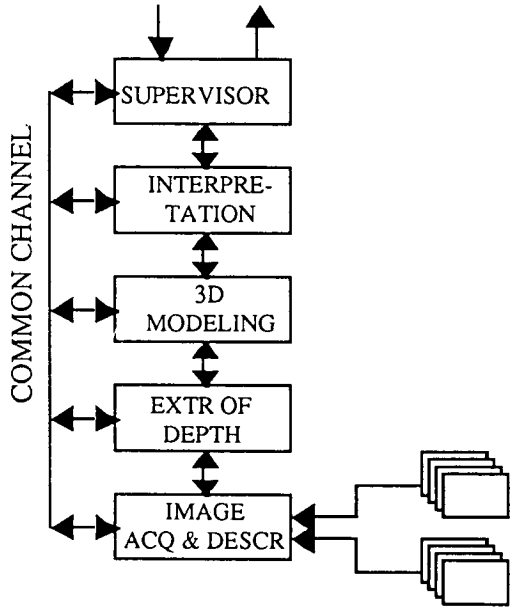


Fig. 3. The VAP system architecture

Having selected a distributed architecture composed of modules, the consortium turned its attention to the design of a common component for each module. At the very least, this standard module must provide the communications interface. It was soon observed that scheduling was a basic to continuous operation and a cyclic scheduler was provided which calls the procedures which implement each phase of computation. The phases of operation included phases for integration of new data and phases for control of processing.

In order to obtain temporal context, the consortium drew on previous results in image tracking. A tracking architecture was defined composed of the phases predict-match-update [Crowley et al. 88], [Granum-Christensen 88] based on techniques used in the control community since the early sixties [Kalman 60]. The architecture is shown in figure 4.

The *analysis* block, in figure 4, is responsible for the frame by frame analysis, which generated a set of geometric primitives (or tokens). The correspondence with information in the temporal context is performed in the *match* block. To simplify matching the information in the temporal context is used in a prediction of the expected content of the next frame. Once correspondence has been established the information contained in the internal models must be updated to reflect the new information contained in the new frame. Once updating has been carried out the cycle may start all over again.

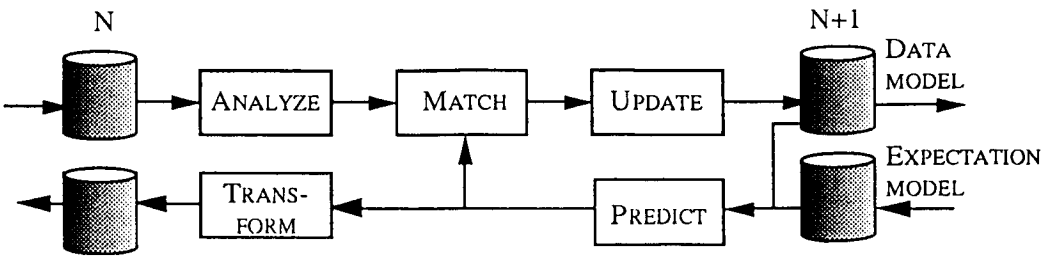


Figure 4. Basic Predict-Match-Update cycle for the module architecture.

As a model at level N+1 is used for prediction of primitives in the next frame, the predictor may also be given other types of input which can be used for guidance of processing. Introduction of goal derived information into the model at level N+1 will consequently allow top-down/attention based control of processing. A prediction may be transformed into a representation that is compatible with the one used the level below, so that it may drive processing at the level below. This flexibility facilitates investigation of different control strategies.

2.5 Control Issues

Construction of an operational system includes issues in control to ensure satisfaction of user defined goals. Goals are widely recognised as a fundamental component of intelligent systems. The consortium initially defined a set of general goal commands :

search(X): Is X present in the scene?

find(X): Where is X, given X has been identified earlier?

relate(X,Y): What is the spatial relation between X and Y?

describe(X,Y): Determine property Y for object X

watch(X): Allocate resources for notification of changes for X

track(X): Maintain a dynamic description of object X.

This set of goals defines the user level interface to the system. Based on the potential goals, the system must be able to allocate its resources for optimal satisfaction of the concurrent goal(s).

A number of approach to goal directed processing have been reported in the vision literature. Most of these efforts include use of a cyclic process, in which data received are matched against expectations. Depending on success or failure in the matching process an updating or event detection process is used to drive the next cycle of processing. An example of such a cyclic process is described by [Tsotsos 87] for the ALVEN system.

When the VAP effort was initiated the use of production systems and reasoning under uncertainty appeared the most promising in terms of providing insight into the problem of the cycle of planning, sensing and interpretation. These tools have been incorporated into the system. In system level control, externally defined goal commands are translated into actions by rule based planning. Planning generates the sequence of state transitions (actions and their parameters) expected to allow completion of a goal. These actions are then executed by one or more system modules. The internal handling of such actions is an issue that is resolved for each of the modules. A skeleton system constructed by the consortium provides the framework for experiments in control and coordination of visual processes.

3. The SAVA III Skeleton System

In order to perform experiments in control and integration of a continuously operating vision system, the VAP consortium constructed an empty "skeleton" system. This skeleton was then provided to partners so that they could "fill in" the functional parts needed for their experiments[2]. This system was named SAVA, for the french acronym "Squelette d'Application pour la Vision Active". The SAVA Skeleton system provides a standard module with communication and interface components that permit an experimenter to construct and run distributed real time vision system.

The structure of SAVA has evolved with our understanding of the problems of integration and control. The original SAVA system was released at month 12 of VAP-I. Experiences during the second year of VAP-I brought out a number of shortcomings in the design. A team composed of people from AUC, KTH and LIFIA designed a revised version, SAVA II, which was released at the month 24 milestone. An intense integration effort was performed in preparation of the month 33 integrated demonstration, with software and hardware contributions from all VAP partners integrated into SAVA II. Modifications in communications and interface design, as well as a large number of small improvements, led to release of SAVA 2.4 in March 1992.

Experience with SAVA II has shown the importance of demons for combining purposive and event driven control of perception. This led to the desire for an interpreter for demon functions. In addition,

[2]The incompatibility of successive releases of MOTIF have created problems for portability and have cost the consortium considerable time and money.

programming control experiments in SAVA II was a somewhat difficult task. Control knowledge was embedded in procedural code and thus hard to understand or change. It was decided to design a control system based on an interpreter for declarative expressions of the control logic. From these two needs emerged the idea of using the CLIPS 5.1 rule interpreter for the control component and the demon interpreter within each module. CLIPS 5.1 is written in C and is provided with the full source code. As a result, it was extremely easy to integrate the SAVA modules into the CLIPS environment.

A new version of the skeleton system, SAVA III, has been created based on the principle of interpreting control information. In SAVA III, most of the procedures for processing and communication are written as C procedures and explicitly declared to the CLIPS 5.1 rule interpreter. Rules and functions are then written using these procedures. The basic processing cycle is built as a sequence of states with transitions managed by rules. The processing performed within a state can be easily changed based on either perceptual events or external commands. Because the control rules are interpreted, the control sequence for a module may be changed dynamically, without re-compiling a module. It is even possible for a module to send another module function definitions as ASCII messages, using the CLIPS deffunction facility. The rule based scheduler is particularly useful for the implementation of demons. Demons may be programmed as rules which react to the contents of the model as well as to external messages.

In addition to the changes in the control part, a major effort has been made to add the possibility of synchronised operation to the modules. SAVA is a soft real-time system, distributed over a set of workstations operating under UNIX. At some time in the future, we intend to port SAVA onto dedicated hardware running under a real time programming environment. However such systems are relatively difficult to program and debug. The use of UNIX and distributed processing permits the VAP-II project to perform experiments with a reasonable effort.

A synchronisation system has been built into SAVA III in order to compensate for uncertainties in communication and execution time for distributed modules. A synchronisation module provides other modules with a universal time reference. In this way, all information that is processed or communicated is time-stamped, permitting an estimate of dynamic processes to be observed or controlled.

The following sections present a detailed description of the components of the SAVA III system. It first gives a brief overview of the components of the skeleton system and its standard module. It then describes processes for interpreting messages using a rule based interpreter, and the design of "demon" processes that perform pre-attentive detection of events. A description of the rule based control of a module is presented, followed by a description of the synchronisation of modules.

3.1 Overview of the SAVA III Software Skeleton

The SAVA skeleton system is composed of the following components:

1) A launcher program that permits the user to assign modules to processors and to initiate operation.

2) A distributed mailbox system that is launched on the different processors to establish a communications system and to launch the component processes.

3) A library of communication procedures for modules. This library include procedures for communication by message as well as procedures for dedicated high band-width communication between processes.

4) A skeleton module structure built around a scheduler.

5) A set of graphical man-machine interfaces.

The SAVA system provides mailbox communication for data, control and acknowledgements, as well as a procedures for dedicated high-band-width channels between modules. Messages include formatting information that permits the message passing system to pack and unpack messages.

Visual perception is performed within processes imbedded in copies of the SAVA "standard module". The SAVA III standard module is shown in Figure 5. The standard module is composed of a number of procedures (shown as rectangles) that are called in sequence by a scheduling process.

A SAVA module repeatedly executes a cycle in which it:

1) Acquires new data.
2) Transforms this data into an internal representation.
3) Makes predictions from its internal model.
4) Matches the predictions with the transformed data.
5) Uses the match results to update the internal model.
6) Executes demons to detect perceptual events within the internal model.

This cyclic process is executed by a rule-interpreter. Each phase corresponds to a state in which a particular operation is performed. At each state transition new messages which have arrived on the mail box channel are read and processed. Such messages may change the procedures that are used in the process, change the parameters that are used by the procedures, or interrogate the current contents of the description that is being maintained.

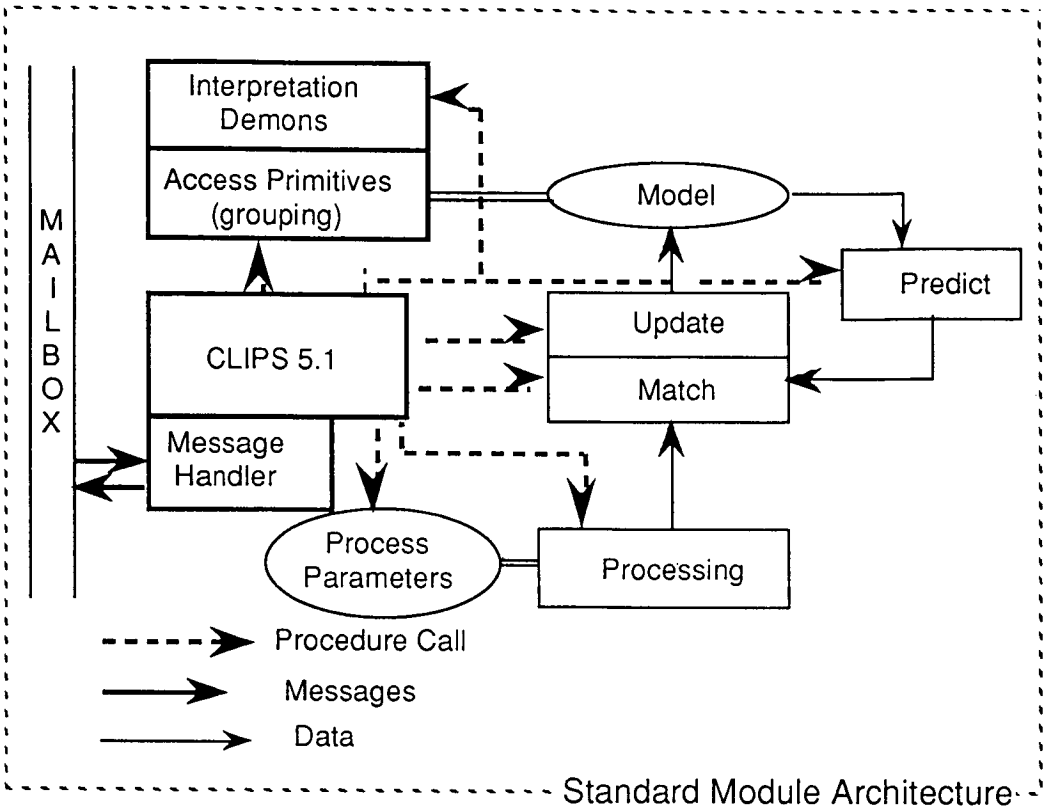


Figure 5 Architecture of a Standard Module in SAVA III

The cyclic process within a module is managed by a control token placed on the working memory. This control token is a simple list in which the first atom is the word "phase" and the second is the name of one of the phases: {get-data, transform, predict, match, update, messages, demons}. The definition of theses phases is as follows:

get-data Acquire a new observation

transform Transform the data to the internal representation

predict Predict the contents of the observation

match Match the prediction to the observation

update Update the model using the correspondence of the prediction and the observation

demons Execute a set of automatic procedures for event detection.

At the end of each cycle, the scheduler executes a set of demons. Demons are responsible for event detection, and play a critical role in the control of reasoning. Some of the demon procedures, such as motion detection, operate by default, and may be explicitly disabled. Most of the demons, however, are specifically designed for detecting certain types of structures. These demons are armed or disarmed by recognition procedures in the interpretation module according to the current interpretation context.

In the SAVA III system, the procedures of a module are made explicitly available to the CLIPS 5.1 rule based interpreter. This includes the original SAVA II scheduler, so that the system is upwards compatible. In addition to acting as a scheduler, the rule interpreter is also used to define the control part of demons and to interpret messages from other modules.

3.2 Communications Between Modules

Modules communicate control, data requests, reply and synchronisation information using message passing based on Unix Sockets. The SavaSend() function is used to send mailbox messages to other modules. A SavaSend command contains three fields:

Header: The destination and type of message.

Format: An ASCII description of the message format.

Body: The message including commands and parameters.

The destination is a symbolic name for another module. The types of message may be control, acknowledge or data. All message exchanges are initiated by a control message. The format string is transmitted with the message and is used both to encode and decode the message. In this way a change in message protocol may be made with a minimum of difficulty. This format string can contain conversion directives like %d, %f, and %c, based on the C language printf protocol. We have added conversion

directives for sending arrays, structures and images.

Many SAVA functions accept a variable number of arguments. Furthermore, the type of these arguments is unspecified. These functions accept a fixed number of normal arguments, followed by an arbitrary number of arguments of unknown type. The last normal argument is a format string which describes the arguments following it.

Large data structures may be communicated between modules using dedicated sockets. Communication of dedicated channels are performed by the functions SavaRead and SavaWrite. As with mail-box, messages, high band-width channel messages are encoded with an ASCII format directive which is transmitted with the message. High band-width channels in SAVA are faster than message passing because the channels provide a direct connection. No intermediate routing is necessary.

Messages passed through the mail box communication system are interpreted by the rule interpreter. New messages are transformed into working memory elements by the function "checkmessage". Checkmessage creates a list in working memory composed of the keyword "message" followed by the name of the sender, a message keyword and and ASCII string with the body of the message. Checkmessage assures upwards compatibility with message types that were defined in SAVA II which have not be transformed to SAVA III.

The checkmessage function is executed at the end of each phase of the standard module. For example, the transition from match to update is performed by the rule "update-phase" :

```
(defrule update-phase
      (declare (salience -100))
      ?p <- (phase match ?c)
=>
      (check-message)
      (retract ?p)
      (assert (phase update ?c))
)
```

The result of check-message is a list of the form:

```
(message  ?sender ?command ?body)
```

If ?command string is tested to determine how the message should be interpreted. If command corresponds to one of the CLIPS keywords "deffunction" or "defrule", then ?body is interpreted by the CLIPS function BUILD. This is permits external modules to define functions and rules using the CLIPS deffunction and defrule constructs.

If ?command corresponds to a previously defined function, then ?body is executed using the clips "eval" construct. If ?command is unknown, or the interpretation is not successful, eval returns FALSE. The result returned by eval is used by the function "reply" to send a reply to the sender. If the message evaluates to a "NIL", then reply does not send a message.

The CLIPS function "build" will interpret a string as if it has been typed to the interpreter. This may be used to interpret defrule and deffunction messages from other modules, as shown by the rule "interpret-def-commands". The command "mv-append" is used to compose a list with the desired commands.

```
(defrule interpret-def-commands
      (declare (salience 100))
      ?m <- (message ?sender ?command ?body)
      (test (member ?command
            (mv-append deffunction defrule)))
=>
      (reply ?sender (build ?body))
      (retract ?m)
)
```

Functions may be defined at initialisation or by messages from other modules. A function, encoded in an ASCII string, may be executed using the CLIPS "eval" command, as shown by the rule "interpret-function-messages"

```
(defrule interpret-function-message
      (declare (salience 100))
      ?m <- (message ?sender ?command ?body)
      (test (member ?command
            (mv-append list-deffunctions))
=>
      (reply ?sender (eval ?body))
      (remove ?m)
)
```

If the interpretation is not successful, eval returns FALSE. Unless the result is NIL, it is sent to the ?sender in a reply message.

3.3 Automatic Interpretation by Demon Processes

A demon is an automatic procedure which operates on the internal model of each module to detect events. Currently active demon procedures are executed after the update phase of each cycle. Demons are responsible for event detection, and play a critical role in the control of reasoning. Some of the demon procedures, such as motion detection, operate by default, and may be explicitly disabled. Most of the demons, however, are specifically designed for detecting certain types of structures.

Demons may be invoked by other demons or by commands received from other modules, including from a human supervisor. A demon is instantiated by entering a demon token in working memory. A demon token is simply a list with three elements:

```
(demon  <name> <id>)
```

where <name> is the name of the demon and <id> is a unique identity determined by the function "gensym". Multiple copies of the same demon can be instantiated, each having its own "id". Each demon can create its own state in working memory, indexed by <id>. A demon can be removed by removing the demon token

from working memory.

The control part of a demon is encoded as rules. As an example consider a demon to find ellipses in the image:

```
(defrule ellipse-finder   "The demon for an
ellipse finder"
      (phase demons)
      (demon ellipse-finder ?id)
      (ellipse-demon-data  (id ?id)
                  (parameters ?p))
=>
      (assert (get-ellipses ?p)))
```

If we suppose that the function "get-ellipses" will instantiate a structure of type ellipse for each ellipse found, then a second rule can be written to treat each ellipse.

```
(deffrule  hypothesize-cylinder "generate
cylinder hypotheses"
      (phase demons)
      (demon cylinder-finder)
      (ellipse (id ?id) (cx ?x) (cy ?y)
            (major ?ma) (minor ?mi)
            (angle ?angle))
      (test (< 5  (abs ?angle)))
=>
      (assert (cylinder  (cx ?x) (bottom ?y)
            (radius ?ma) (ellipse ?id)))
      (assert (cylinder  (cx ?x) (top ?y)
            (radius ?ma) (ellipse ?id))))
```

Other rules can be used to detect the existence of cylinders with the same axis and to reduce cylinder hypotheses to a minimum number, or to use the hypothesis of a cylinder with several ellipses to generate the hypothesis of a cup.

Goals for the module can be entered into working memory as a three element list:

```
(goal  <name>  <priority>)
```

Goals can then have the effect of activating and dis-activating demons. An example of a goal invoking a demon is the rule "cup-demons".

```
(deffrule  cup-demons
      "invoke the cylinder finder"
      (phase demons)
      (goal find-cup ?p)
=>
      (assert (demon cylinder-finder)))
```

An example of removal of demons is the rule "remove-non-cup-demons".

```
(deffrule  remove-non-cup-demons
      "remove cup demons"
      (phase demons)
      (goal find-cup ?p)
 ?d <-(demon ?n ?id)
      (not (?n cylinder-finder))
=>
      (retract ?d))
```

Having a rule interpreter provides explicit control knowledge for demons and their control logic. It also permits the working memory to be used to create and free working memory for representing demons state. The result is a flexible, easy to use, tool for experiments in control of perception. In the following section we present an example of such control.

4 The Visual Navigation Demonstrator

This section illustrates the use of SAVA III by presenting an overview of the a visual navigation system. This system was constructed for the milestone 1 demonstration of VAP-II presented in June 1993. The structure of the demonstration system is shown in figure 6. The system is composed of processes for

1) Fixation control of the binocular head.
2) Local navigation actions for a mobile robot.
3) Image acquisition and processing.
4) Tracking and grouping a 2-D description of the contents of the image.
5) Computing and maintaining a 3-D description around a fixation point.
6) Recognition of landmarks and object.
7) System Supervisor for coordinating processing of the other system modules.

Fixation Control Unit

The fixation control unit provides a standard interface to the device controller for the VAP/SAVA binocular stereo head. This module maintains a copy of the current state of the fixation point and the component axes for the binocular head. It receives commands in the form of tasks expressed in either device or motor coordinates. Commands are communicated to the binocular head, the robot-arm (neck) or the mobile platform using such device-level control.

The fixation control unit also contains facilities for programming procedural style "perceptual actions". Such perceptual actions are reflex procedures that command the state of the binocular head at either the device level or the motor level based on measurements made from images. Examples of low level perceptual actions include ocular reflexes for servoing aperture, focus and vergence. Other examples include procedures for tracking a moving object.

Image Acquisition and Processing

The image acquisition and processing module handles all image processing requirements for the other modules, thus minimizing the communication requirements. This module is based on two computer cards constructed by the consortium. The first of these, the Pyramid card digitizes synchronised stereo images and immediately computes a 12 level binomial pyramid for the two images. Processing time for each pair of images is 40 ms. The second card extracts edge segments using Gaussian derivatives.

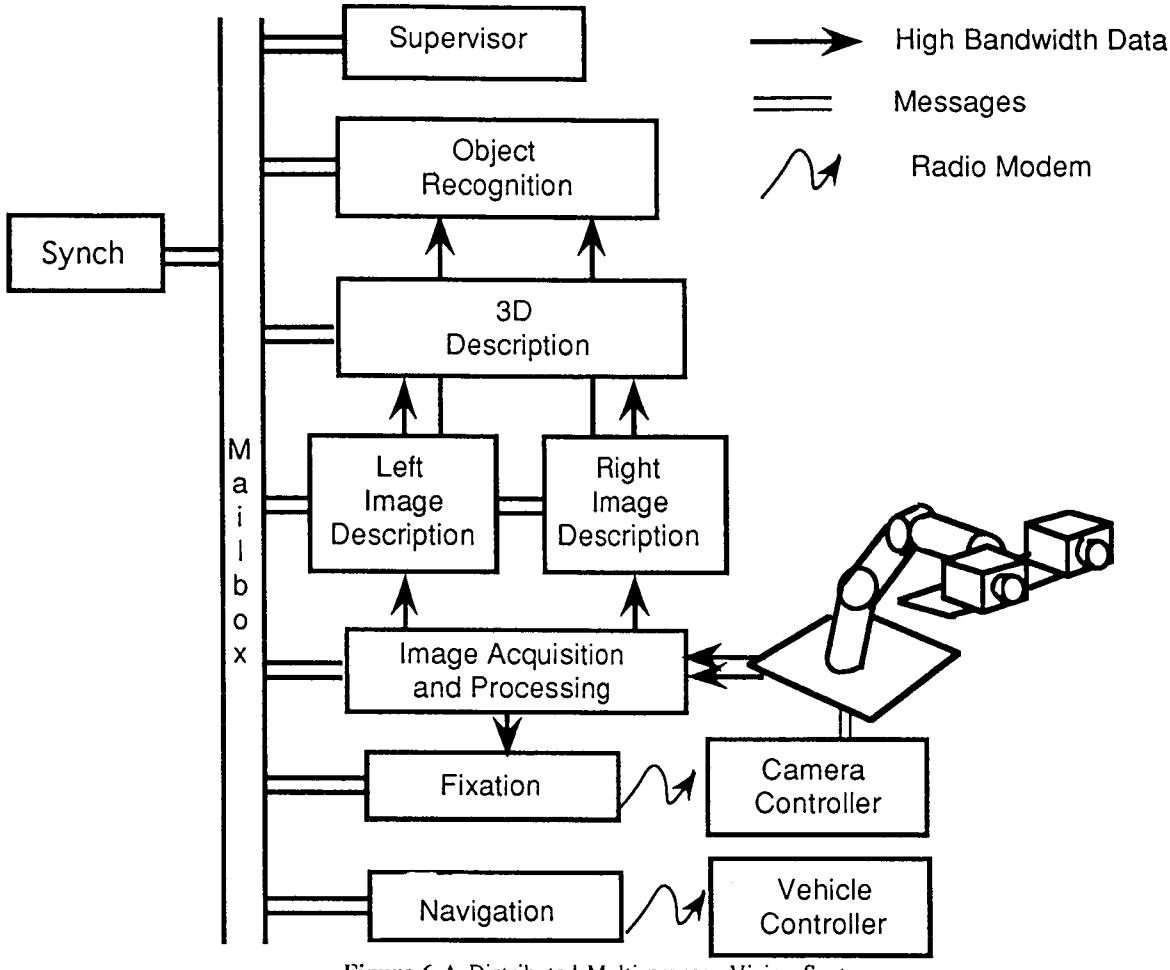


Figure 6 A Distributed Multi-process Vision System.

The edge extraction process begins by calculating the horizontal and vertical derivatives within the region of interest. These derivatives are then combined by table look-up to compute the gradient magnitude and orientation. Points which are extrema in magnitude are marked as potential edge points and compared against two thresholds. Hysteresis thresholding is applied so that only regions of edge points containing at least one point above the threshold are considered. Adjacent edge points with a similar orientation are grouped to form line segments. Edge segments are represented by a vector of parameters that includes the mid-point, orientation and half-length.

Three classes of image processing procedures are available in the image processing module

1) Edge Segment Extraction. On command, the module will transfer the pixels within a region of interest to an edge extraction card produced by the consortium. This card computes the gradient magnitude and orientation and detects pixels which are extrema in gradient magnitude. Detected pixels are grouped in single raster scan to construct edge segments. Gradient magnitudes are compared to two thresholds to provide a hysteresis based thresholding.

2) Edge Chain Extraction: In place of edge segments, another module may request edge chains. Edge points are are computed by the same algorithm as for edge segments. A one pass raster-chaining algorithm is used to construct a list of edge chains within the region of interest. The edge chaining code is computed on a co-processor card based on the Intel 680

3) Measures for Ocular Reflexes. In order to avoid communicating images, the measurements on which ocular-motor reflexes are based have been placed in this module. Measures include coarse to fine computation of phase for convergence, and gradient based measurements for aperture and focus.

Image Tracking and Description

An image description is maintained by a tracking process which uses a first order Kalman filter to track edge segments. This tracking process improves the stability of image primitives, permits the system to maintain correspondence of image features over time, and provides an estimate of the position and velocity of image primitives as well as the uncertainty of these

estimates. It also permits information about the movement of the head or vehicle to be used to compensate for movements by the robot. A vocabulary of model access and grouping procedures give associative access to the 2D description modules. These procedures are used by a library of "demon" procedures which can be enabled in order to provide data driven interpretation of the image description.

Separate image description modules exist for the right and left cameras. The 2D image descriptions are maintained by a tracking process that uses a first order Kalman filter to track image description primitives. This tracking process improves the stability of image primitives, permits the system to maintain correspondence of image features over time, and provides an estimate of the position and velocity of image primitives as well as the uncertainty of these estimates.

Model access primitives use matching and grouping to interrogate the contents of the token model. A set of demons may be invoked by other modules to interrogate the description after each update using the model access primitives. Access to the 2D model is provided by a large vocabulary of model access and grouping procedures. It is also possible to compose sequences of these grouping procedures, extracting, for example, all the junctions near an ideal line. These procedures may be called by other modules within the system, or they may be invoked by a set of interpretation demons. These demons are placed on an agenda by messages from other modules. After each update cycle the demon agenda is executed.

3D Geometric Scene Description Module

In addition to a description of images, the skeleton system maintains a geometric description of the scene. This geometric description expresses the structure within a region of interest of the scene in terms of 3D parametric primitives. This module assumes that the phase based convergence reflex maintains the cameras converged on an object. Convergence maintains edge segments from a region of interest in the scene in the similar positions in the image. The image description access primitive "FindPrototypeSegment" is used to construct a list of possible matching segments in the left and right image. This list is sorted based on similarity of length, orientation and position. The most likely matches are selected for 3D reconstruction.

Reconstruction requires camera calibration. A novel procedure for dynamic auto-calibration of cameras has been developed. This procedure permits a reference frame for a pair of stereo cameras to be constructed for any scene objects. The projective transformation matrices from object centered coordinates can be obtained by direct observation (no matrix inversion) and can be maintained by a very simply operation. These matrices make it possible to reconstruct the 3D form of objects in an object centered reference frame. As with the image tracking and description module, the geometric description is maintained by a tracking process in order to provide stability and to maintain correspondence over time.

Symbolic Scene Interpretation

The symbolic scene interpretation maintains a symbolic description of the scene in terms of known object categories (or classes) and qualitative relation. This description is built up and maintained by interrogating the contents of the image and scene description modules. The SAVA III symbolic description process was implemented using the CLIPS rule interpreter system. Rules implement a hypothesize and test process which is triggered by demons. Working memory of the production system serves as a blackboard into which recognition procedures can poste their results.

Process Supervisor

The process supervisor maintained a list of places and routes which the system is to travel, as well as a data base of "landmarks" which the system is to find during mission execution. The supervisor plans a navigation which it then executes by sending commands to the other modules. An interesting aspect of the supervisors operation was coordinating between the competing tasks of watching in front of the robot for obstacles and searching for landmarks for position correction. Obstacles must be searched at least once every 50 cm, while landmark detection is required whenever the uncertainty of the estimated position passes a certain threshold. Both operations require command of the camera head. This balancing act was performed by a finite state automata programmed as a set of rules.

Navigation Control

The navigation module controls vehicle actions by sending commands to an on-board vehicle control program. The on-board program, known as the "standard vehicle controller", provides asynchronous independent control of forward translation and rotation. The on-board controller acts like auto-pilot, stabilizing the vehicle and estimating its position. The controller accepts both velocity and displacement commands, and can support a diverse variety of navigation techniques. The controller is capable of responding to commands at any time using a simple serial line protocol. New commands for displacement immediately replace previous commands. This permits visual servoing to be used to pilot the vehicle.

Position and orientation are modelled in the vehicle controlled using Kalman filter to maintain an estimated position and covariance. The control protocol includes a command to correct the estimated position and orientation and their uncertainty from external perception using Kalman filter update. This command has been used to update the estimated position by observing the angle to known objects. The LIFIA standard vehicle controller is described in greater detail in [Crowley-Reignier 93]. The navigation module contains procedures to detect and avoid obstacles, and to locate and use landmarks for updating the vehicle's estimated position.

5. Conclusions

According to the principle of "purposiveness" a vision system operates in order to furnish an observation function for some task. In order to enrich our task domain, we have adapted the VAP Skeleton system to serve as the visual component for a mobile robot navigating in an indoor environment. We stress that the visual navigation is not, in itself, the goal of the project. Visual navigation is a task which is sufficiently rich in events to explore the problems of integration and control of an active perception system.

During the last four years, the VAP consortium has constructed a number of demonstrations of continuously operating vision systems. In each of these systems explicit control of sensor motion and processing has permitted the system to operate in real time, with increasingly degrees of robustness. The consortium experience has verified the VAP hypothesis that control of continuously operating process is basic to the design of a general purpose real time vision system.

Bibliography

[Aloimonos et. al. 88] Aloimonos, J. Y., I. Weiss, and A. Bandyopadhyay, "Active Vision", International Journal of Computer Vision, Vol. 1, No. 4, Jan. 1988.

[Bajcsy 88] R. Bajcsy, "Active Perception", IEEE Proceedings, Vol 76, No 8, pp. 996-1006, August 1988.

[Ballard 91] D. Ballard, "Animate Vision", Artificial Intelligence, Vol 48, No. 1, pp. 1-27, February 1991.

[Bolles-Horaud 84] Bolles, R. C. and Horaud, P., "Configuration Understanding in Range Data", Second ISRR, August, 1984.

[Brooks 81] R.A. Brooks, "Symbolic Reasoning among 3-D models and 2-D images", Artificial Intelligence, Vol. 17, pp. 285-348, 1981.

[Brown 90] C. Brown, Prediction and Co-operation in Gaze Control, Biological Cybernetics 63, 1990.

[Crowley 87] Crowley, J. L., "Coordination of Action and Perception in a Surveillance Robot", IEEE Expert, Vol 2(4), pp 32-43 Winter 1987. (Also in the 10th I.J.C.A.I., Milan 1987).

[Crowley et. al. 88] J. L. Crowley, P. Stelmaszyk and C. Discours, "Measuring Image Flow by Tracking Edge Lines", Second I.C.C.V., Tarbon Springs, Fla, Dec. 1988.

[Crowley 91] Crowley, J. L. "Towards Continuously Operating Integrated Vision Systems for Robotics Applications", SCIA-91, Seventh Scandinavian Conference on Image Analysis, Aalborg, August 91.

[Crowley-Reignier 93] J. L. Crowley et Patrick Reignier, "Asynchronous Control of Rotation and Translation for a Robot Vehicle", Robotics and Autonmous Systems, Vol 10, No. 1, January 1993.

[Crowley-Christensen 93] J. L. Crowley and H. I. Christensen, Vision as Process, Springer-Verlag Basic Research Series, To appear, 1993.

[Draper et al. 89] B. A. Draper, R. T. Collins, J.Brolio, A. R. Hansen, and E. M. Riseman, "The Schema System", International Journal of Computer Vision, Kluwer, 2(3), Jan 1989.

[Eklundh 92] Eklundh, J. O. and K.Pahlavan, Head, "Eye and Head-Eye System", SPIE Applications of AI X: Machine Vision and Robotics, Orlando, April 92.

[Granum-Christensen 1990] E. Granum & H.I. Christensen, Dynamic Robot Vision, In: Traditional and Non-traditional Robotics Sensors, T. Henderson (Ed.), NATO ASI Series in Computer Science, Springer Verlag, 1990.

[Hanson-Riseman 1978] Hanson, A.R. & Riseman, E.M., VISIONS: A Computer Vision System for Interpreting Scenes, in Computer Vision Systems, A.R. Hanson & E.M. Riseman, Academic Press, New York, N.Y., pp. 303-334, 1978.

[Huang 83] Huang T. H., Image Sequence Processing and Dynamic Scene Analysis, Springer Verlag, Berlin, 1983.

[Kalman 60] Kalman, R. E. "A New Approach to Linear Filtering and Prediction Problems", Transactions of the ASME, Series D. J. Basic Eng., Vol 82, 1960.

[Krotkov 90] Krotkow, E., Henriksen, K. and Kories, R., "Stereo Ranging from Verging Cameras", *IEEE Trans on PAMI*, Vol 12, No. 12, pp. 1200-1205, December 1990.

[Marr 82] Marr, D., Vision, W. H. Freeman, San Francisco, 1982.

[Tsotsos 87] Tsotsos, J. "Image Understanding", The Encyclopedia of Artificial Intelligence, S. C. Shapiro (Ed), Wiley, New York, 1987.

[Tsotsos 88] Tsotsos, J. "A Complexity Level Analysis of Computer Vision", International Journal of Computer Vision, 1(4), Jun 1988.

[Weems 89] C. Weems, "The Image Understanding Architecture", International Journal of Computer Vision, Kluwer, 2(3), Jan 1989.

AN ARCHITECTURE FOR REAL-TIME VISION PROCESSING

Chiun-Hong Chien

Intelligent Systems Department
Lockheed Engineering and Sciences Company
2400 NASA Road 1, Houston TX 77058
chien@superman.jsc.nasa.gov

Abstract

This paper proposes an architecture for real time vision processing on parallel processors with physically distributed shared memory, and presents an initial implementation of the architecture on i860-based Mercury Computing Systems. Within the framework of the architecture, each vision function (such as median filtering or contour extraction) is defined as a task or a set of tasks. A collection of these tasks, along with associated data, may be recursively divided into subtasks and processed by multiple processors through the coordination of a task queue server. The task queue server resides in shared memory accessible by all the processors. Each idle processor subsequently fetches a task and associated data from the task queue server for processing and posts the result to the shared memory for later use. In this way load balancing within the parallel processing system can be achieved without a centralized controller, as demonstrated by experimental results.

1. Introduction

It is well known that vision processing involves a tremendous amount of computation. It is even more so for real time vision processing such as vision guided grasping of free-floating objects in space [1], in which processing cannot be carried out in real time without high processing power provided by parallel computers such as Hypercubes [2] or i860-based Mercury Computing Systems [3]. Even with the availability of powerful parallel computers, the real challenge is to find the best strategies for mapping data and vision tasks onto underlying parallel architectures.

Copyright ©1993 by the American Institute of Aeronautics and Astronautics, Inc. No copyright is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all rights under the copyright clamed herin for Governmental purposes. All other rights are reserved by the copyright owner.

A great deal of effort has been directed toward exploring parallelism in pixel-level image processing by taking advantage of its simplicity and data regularity [4]. The success of parallel image processing, however, has not been extended to mid-level feature processing or high-level image understanding. This is due to:

1. simple data partitioning methods do not fit to the mid-level and high-level vision processing, and
2. existing Ethernet-based network discourages dynamic data/task migration.

The advance of VLSI technology in the past decade makes it possible to build tightly-coupled parallel computers with powerful general-purpose microprocessors (such as i860s) interconnected by high bandwidth crossbar switches. Furthermore, research in physically distributed shared memory [5] has reached a stage where the support of physically distributed shared memory model on commercial parallel processing systems becomes a standard, rather an exception. The availability of a powerful parallel processing system with the support of the physically distributed shared memory model has encouraged us to study more powerful methods for data and task partitioning, task allocation, task scheduling, and load balancing, in mid-level feature processing and high-level object recognition and pose estimation. The result of this effort is the design of our proposed architecture for real time vision processing.

The proposed vision architecture has evolved from a vision architecture, known as PARADIGM [6], designed and implemented on NECTAR (a fiber-optics based high-speed network backplane for heterogeneous multi-computers) [7]. PARADIGM was designed to provide mechanisms and primitives that not only allow vision-related parallel programs to be developed with ease, but also maximize program concurrency at both task-level and subtask-level. While developing their programs, users need only focus efforts on problem solving and task partitioning, without

a need to worry about the details of communication and task scheduling. PARADIGM is a distributed system with centralized control. It is composed of a controller and a number of workers which communicate through message passing.

However, with the support of physically distributed shared memory, it deems unnecessary to have a controller for task management (e.g. distribution and scheduling), and it would be less efficient for workers to "communicate" with each other through message passing. Instead of a controller and several communication servers, a task queue server is used both for "coordinating" task execution and for exchanging information. A task queue server is actually a collection of various queues residing in a shared memory buffer accessible by all the processors/workers. Each idle worker subsequently fetches a task and associated data from the task-queue server for processing and posts results to the shared memory buffer for later use. In this way, the proposed architecture is similar to the blackboard architecture employed in Carnegie Mellon's autonomous land vehicle, Navlab [8].

To study the feasibility of using the proposed architecture for real time vision processing, the task queue server has been implemented on an i860-based MC860VS [2]. Parallel algorithms for median filtering and contour extraction have also been designed and implemented on the MC860VS. Timing statistics has been collected and analyzed in order to measure the overhead for task management and to refine the proposed architecture.

The remainder of this paper is organized as follows. Section 2 gives a brief description of the MC860VS. Discussed in Section 3 are the criteria considered in the design of the proposed architecture. Section 4 presents the design of, and functionality provided in the proposed architecture, which is followed by experimental results from an initial implementation in Section 5. Concluding remarks given in Section 6.

2. Mercury Computing Systems MC860

A MC860 is i860-based array processor (AP). A MC860VS board contains four computing elements interconnected by a six-port crossbar switch. Each computing element consists of a 40 MHz i860 processor, a high-performance data switch, a DMA controller, up and to 16 MBytes DRAM (as shown in Figure 1). The MC860 acts as a peripheral to a host computer such as a Sparcstation or a 68040-based vxWorks platform [3].

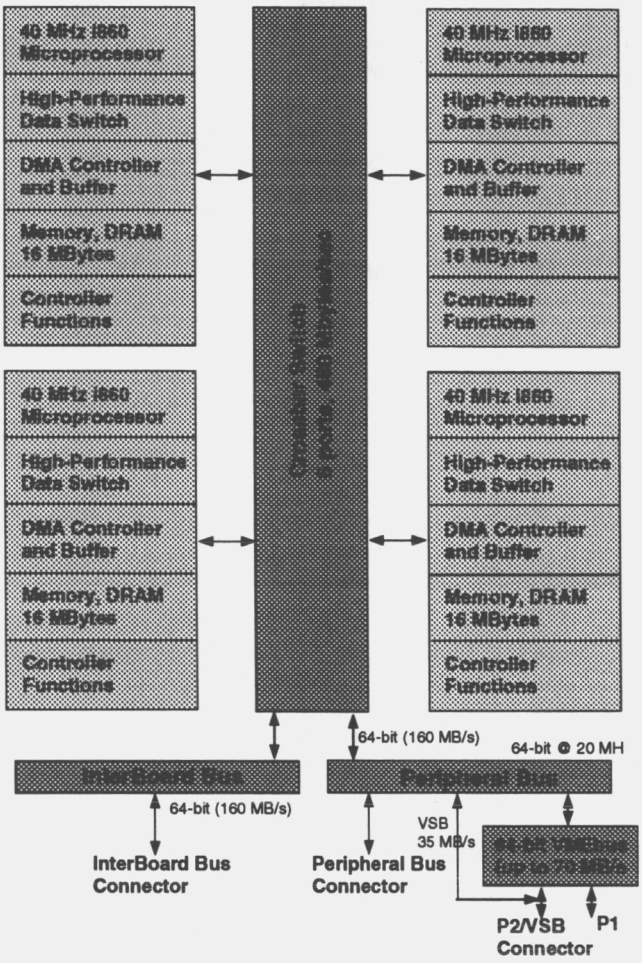


Figure 1: The MC860VS block diagram

The characteristics of computing elements includes

1. Eighty (Sixty) megaflop of computational power for single (double) precision operations.
2. One hundred and sixty megabytes per second transfer rate to AP memory (through a high performance data switch).
3. A 640 Mbytes/sec internal data transfer rate.
4. A 4 Kbytes instruction cache and an 8 Kbytes data cache.
5. A shared memory interface between the host and the MC860.
6. A DMA controller that allows for a memory to memory transfer rate of 160 Mbytes/sec without the involvement of the CPU.

The software supports for MC860s includes

1. An operating system (MC/OS) supports synchronization features such as semaphores and sockets, and others such as multi-tasking, various types of interrupts, and mapping of external memory, among others. It also supports a physically distributed shared memory model for ease of parallel programming.

2. A Scientific Algorithm Library (SAL) and its extended version (ESAL) consist of hundreds of micro-coded primitive functions designed to provide fast memory throughput (and processing speed) by utilizing the data cache (via indirect or direct access) as an extended registers.
3. A rich software development environment in which the user can develop an application in either of three ways as follows:
   - The Application Accelerator (Transparent) approach.
   - The Subroutine Engine approach.
   - The Multicomputer (Attached Processor) approach.

The Transparent approach allows quick testing. The Subroutine Engine approach is suitable for SIMD types of parallel processing. The Multicomputer approach is feasible for MIMD types of parallel processing, and is the approach used for the implementation of our proposed architecture.

3. Design Consideration

This section describes criteria considered in the design of the proposed architecture. We first identify the characteristics of different vision tasks. Vision processing can be roughly divided into three levels: low-level image processing, mid-level feature processing, and high-level image understanding. Their characteristics are as follows:

Low-level Image Processing:

Data items are pixels, which are uniformly distributed in the image space. Operations include simple local or neighborhood operations (e.g. thresholding, filtering, and edge detection) performed on a large amount of data. The inherent parallelism is fine-grained at a pixel level.

Mid-level Feature Processing:

Data items are 2D features such as points, lines, and regions of which the distribution in the image space are not uniform. There are a medium number of data items. Relations among data items are spatial relationships such as adjacency, overlapping and containment. Operations on these data items include unary operations for computing geometric properties, and binary operations involving spatial relationships. The potential parallelism is medium-grained at either feature level or at a level of subsets of spatially adjacent features.

High-level Image Understanding:

Data items are natural or cultural objects or subparts of these objects, which are not uniformly distributed in the image space. Operations include matching between objects (and their subparts) and possible models. The number of data items in general is small, but the number of possible models may be large. The potential parallelism is at the task level or at the level of subsets of the solution space.

As reported in the literature [4] parallel image processing has been successfully applied to real systems such as Warp and Connection Machine. A few systems (such as the CMU Navlab system [9]) have also been developed by exploiting task-level parallelism. However, some tasks are inherently time-consuming and may become bottlenecks during processing if only task-level parallelism is exploited. This problem can be overcome by supporting both task-level and sub-task level parallelism, in addition to pixel-level parallelism, in our proposed vision architecture. Moreover, the heterogeneous nature of different tasks/subtasks and the variation in processing time raise some crucial issues such as task allocation/scheduling, data/task migration, and load balancing, that are not encountered in low-level image processing and scientific computing. The problem is further complicated by the need to handle spatial features in vision processing (spatial-oriented operations, in particular) due to the dimensionality of spatial features, their complex data structures and tangled topological relationships.

In the past, most parallel algorithms have been designed for a single operation at a time. The underlying assumption was that parallel algorithms for multiple operations could be obtained by pipelining those for single operations. This argument might be true for a sequence of local operations (such as low-level image processing) where data distribution is more or less similar for all the operations. However, the argument does not hold in general cases where data distribution varies with individual operations. In these cases, the overhead involved in data redistribution must be taken into account, and therefore the best algorithm for a single operation may no longer be the best choice when it is used along with other operations. In other words, an architecture for parallel processing must allow us to optimize the performance of a set of heterogeneous tasks as a whole, rather than the performance of each individual operation.

To maximize performance, many operations (i.e. tasks in the context of the proposed architecture) should be executed concurrently as long as no temporal ordering (i.e. dependency) exists between them. The task-level parallelism can be realized using a task-queue mechanism (implemented as a *Task Queue* and a set of *Subtask Queues* in this work). In the task-queue mechanism, a task queue and subtask queues are used to keep all the task, that are subsequently assigned to (or fetched by) any idle processor/worker. Task dependency can be handled by a resource mechanism and will be discussed later. To prevent potential bottleneck, any time-consuming task must be divided into smaller tasks by either dividing the task into subtasks (*task partitioning*), or dividing the data set into subsets (*data partitioning*), or a combination of both. A mechanism for *task partitioning* not only allows a vision system to achieve good load balance, but also makes it possible for users to design a complex parallel/distributed program in a hierarchical modular fashion. That is, a program can be recursively divided into modules, submodules, and primitive operations.

A straightforward *data partitioning* method is to partition the data set into many subsets and distribute the subsets to each processor using a task queue mechanism. However, in the domain of vision and spatial oriented processing, partitioning the data regardless of their spatial relationship usually results in the scattering of spatially adjacent data items among the processors which increases the overhead in data migration for tasks that involve operations on spatially adjacent data items. The overhead may be reduced by using shared memory, rather than message passing via sockets, for communication. There are other issues involved in partitioning spatial-oriented data. Readers are referred to [6] for a more detailed discussion.

In summary, the proposed vision architecture should be designed to:

1. support both task/subtask-level parallelism,
2. use task/subtask queues for task management,
3. use a resource mechanism, along with multiple subtask queues, for task scheduling,
4. hide the details of communication and task allocation/scheduling from users,
5. employ shared memory for communication,
6. provide a mechanism for composing parallel vision programs.

A detailed description of the task queue server is given in the following section.

4. Task-Queue Server

The proposed real-time vision processing architecture is similar to the black board architecture. A physically distributed shared memory buffer "accessible" by all the processors is used for storing and fetching tasks (for execution). The block diagram of the proposed vision architecture is shown in Figure 2. It consists of a master, a taskqueue server, and a number of workers which communicate with the master and with each other through a physically distributed shared memory buffer. The master has a set of user defined functions for data/task partitioning, and for post processing (such as merging of partial results from workers). Each worker is facilitated with a set of user defined functions for task execution.

The heart of the proposed architecture is a task-queue server residing in the shared memory buffer. It consists of a task queue, an internal task queue, a number of subtask queues (one associated with each worker/processor), and a reply queue. The task queue stores a sequence of tasks to be executed. These tasks may be recursively divided into subtasks (based on functionality) and queued in the internal task queue. For each task in the internal queue, there is a corresponding task handler (in the module operated by the master) responsible for partitioning the task into a set of smaller subtasks (via *data partitioning*). These subtasks are then "evenly" placed into the subtask queues subject to certain constraints, such as the locality of data on which the tasks will be executed or resource requirements. (e.g I/O devices). The constraints are used to determine the *executors* of the tasks. The reply queue is for storing information regarding the completion of subtasks.

In the following, we shall give more detailed description about important functionality provided by the proposed architecture, including task partitioning, task scheduling and allocation and communication.

Task Partitioning

The principal responsibility of each task handler is to partition tasks. To maximize concurrency (and hence performance), a time-consuming task should be partitioned into a number of smaller subtasks so that the load of the task can be distributed equally over the workers. There are two techniques to partition a task: *task partitioning* and *data partitioning*.

With *task partitioning*, a task is partitioned into a number of smaller tasks which are usually of different

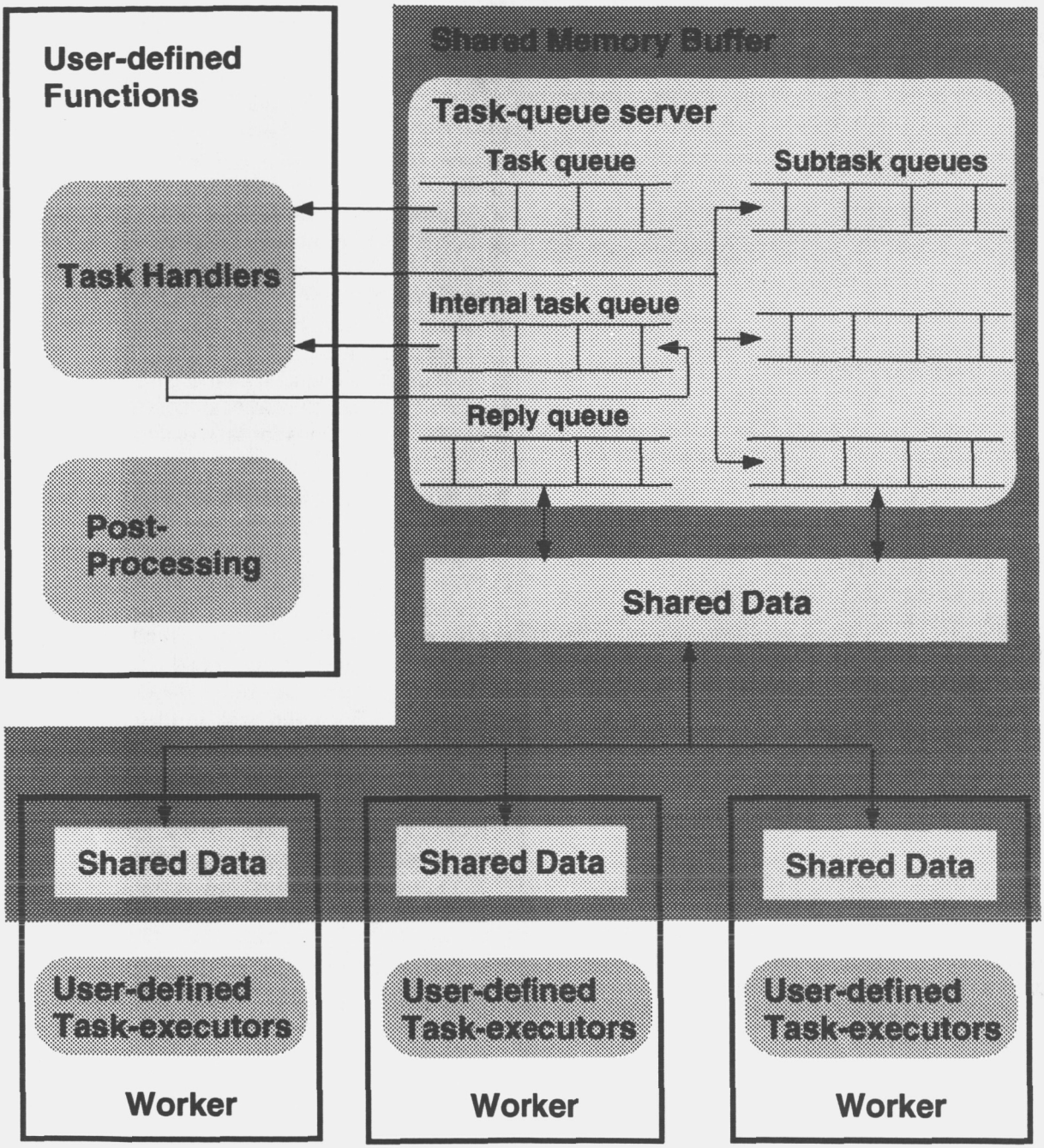


Figure 2: The block digram of the proposed vision architecture

functionality and must be executed in a certain order. For example, a task to extract occluding contours from a noise image may be divided into subtasks for noise reduction, segmentation and contour extraction. The task for contour extraction may not be scheduled until the other two are completed.

Instead of partitioning tasks by functionality, *data partitioning* divides the input data into subsets, which are then processed by workers concurrently and independently. For example, given an image, the task of performing median filtering on the image can be partitioned by dividing the image into subimages, each of which is processed independently by a worker.

To facilitate task and data partitioning, the proposed architecture provides several primitives, including *Create_Task*, *Create_Subtask*, and *Broadcast_Subtask*. The function *Create_Task* creates a new task that is placed in the Internal task queue, waiting to be scheduled. The function *Create_Subtask* creates a

new subtask. A task in the internal task queue may recursively call *Create_Subtask* to create a set of subtasks, that are placed in one or more subtask queues, waiting to be fetched by the associated workers for processing A subtask can be specified as either *worker-dependent* or *worker-independent*. If a subtask is *worker-dependent*, it must be executed by the specified worker. If it is *worker-independent*, it can be executed by any worker, although the specified worker is preferred. A subtask can be marked as *worker-dependent*, when other workers do not have the resource (including data) needed for handling the subtask.

The function *Broadcast_Subtask* allows a subtask to be created and broadcast to all the workers. For example, the task *Task_Exit* is broadcast to all the workers at the end of processing for freeing resources (including memory, sockets, and semaphores).

Task Scheduling

As mentioned earlier, task allocation is carried out by using multiple subtask queues to keep spatial locality, and task scheduling is achieved by using a resource mechanism and a *Depend_On* call. It is important for a distributed system to detect when tasks need competing resources and to schedule tasks to resolve conflicts. In the proposed vision architecture, a resource can be associated with a physical entity (e.g. a display device) or a virtual entity (e.g. data structures). Resources are created with a capacity and tasks can be registered as using a number of resources. A task can be executed only when it needs no resource or the needed resources are available.

By using the resource mechanism properly, synchronization between tasks can often be realized. For example, producer-consumer tasks can be coordinated using the resource mechanism in the following manner. First, the information to be produced by the producer task is registered as a resource of no capacity. When the producer is finished, the capacity of the resource is increased. Therefore, the consumer task, which is registered as needing the resource, will not be scheduled until the producer task is done.

In addition to resource conflicts, there are also task dependencies between different tasks. Task dependency is handled by a function, *Depend_On* (with two tasks as parameters), that constrains a task to be scheduled only when the other task is finished. The function *Depend_On*, together with the resource mechanism, allow task-level synchronization to be specified.

5. Experimental Results

The initial version of the proposed architecture has been implemented on i860-based MC860VS with eight i860 processors, each with 16M memory. In the initial testing, the taskqueue server physically resides on the i860 processor where the master is running. Up to seven workers are running on the same number of i860 processor. The objectives of the initial implementation and testing are as follows:

1. To learn more about the characteristics (i.e strength and limitation) of i860-based MC860VS in the context of real time vision processing.
2. To study the feasibility of using the proposed architecture for real time vision processing.
3. To measure the overhead involved in using the task queue server for parallel processing. The amount of overhead will in turn be a guide line for determining granularity of parallelism used for real-time vision processing.
4. To study the efforts involved in composing parallel programs for vision processing using functions provided by the proposed architecture.

To achieve these objectives, parallel algorithms for several different types of vision operations have been implemented on the MC860VS including algorithms for median filtering and for contour extraction. Timing statistics for running these parallel algorithms on MC860VS have been collected for analysis.

Median filtering is an pixel-level operation, and is easily parallelizable. A typical approach is to divide the image (to be filtered) into N subimages. Each of the N subimages, along with the median filtering operation forms a subtask to be processed by a worker. No explicit merging operation is required when all the subtasks are processed.

On the other hand, contour extraction involves conversion of data structures. i.e. the conversion of a pixel-level representation (image) into a feature or features (contours). One of the approaches to parallel contour extraction is to divide the image into a number of subimages, and to extract a partial contour from each subimage. It is not a "regular" operation in a sense that processing time on each subimage depends on the complexity/shape of the contour in the subimage. It is not a local operation, either, since an explicit merging operation is required to merge the partial contours extracted from the subimages to obtain the complete contour(s).

Experiments for collecting timing statistics were repeated for different numbers of i860 processors (from one to four), for different numbers of subtasks (i.e. 1, 2, 3, 4, 6, 8, 12, 16, and 24, respectively).

Figure 3 shows timing statistics on parallel median filtering (on 256x256 images). For cases where only a single i860 was used, processing time remains relatively the same regardless of the number of subtasks. It implies that the overhead for task management is negligible (when the granularity of parallelism is in the order of tens of milli-seconds) if all the subtasks are processed by a single processor. For cases where two i860s were used, processing time was unusually high (annotated as A2 in Figure 3) when the median filtering operation was divided into three subtasks. This was due to load imbalance. That is, one i860 had one subtask to process while the other had two. The same argument is applied to unusually high processing time annotated as A3 and A4 in Figure 3. It is interesting to point out that processing time at C2 in the figure is slightly higher than B2 and D2. This can be explained as follows. The size of each image on which median filtering was performed is 256x256. The numbers of subtasks associated with B2 and D2 are 8 and 16, respectively. That is, the image was partitioned into subimages of the same size in each of the two cases. Load balance was achieved in these cases. On the other hand, the number of subtasks associated with C2 is 12 by which 256 is not dividable. As a result, load balance was more difficult to achieve since some subimages had larger sizes than the others and took longer time to process.

Load unbalance can also be observed at B3 and D3. The numbers of subtasks in these two cases are 8 and 16, respectively, which are not dividable by 3, the number of processors.

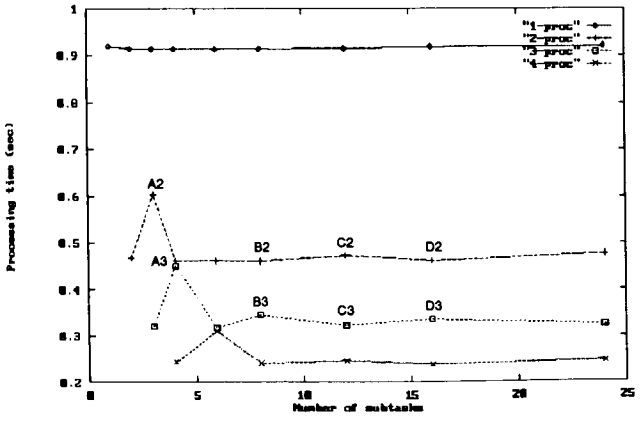


Figure 3: Timing statistics of running parallel median filtering on MC860VS

A different characteristic in timing statistics on parallel contour extraction (as shown in Figure 4) can be observed. For example, in the cases where only a single i860 was used for contour extraction, processing time increased with the number of subtasks. The increase in processing time is not due to the overhead in task management, but due to that in merging the partial contours extracted by each subtask to obtain the complete contour. The overhead is so significant that not much speedup could be obtained by increasing the number of processors beyond three. For the cases where the number of processors is three (or four), the processing time decreases, reaches a minimum, and then increases as the number of subtasks increases. This is due to the fact that a large number of subtasks (running on a relatively small number of processors) will smooth out variation in (and so decrease) processing time for extracting partial contours, but increase time for the merging operation.

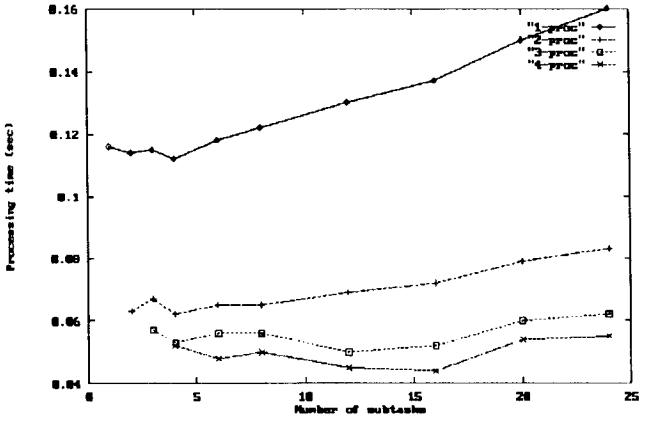


Figure 4: Timing statistics of running paralle contour extraction on up to four MC860VS

To investigate the overhead due to task management in a multi-processor environment, another experiment was conducted to measure speedup factors by running parallel median filtering on one to seven i860 processors. In this experiment, the number of subtasks was set equal to the numbers of processors so that load imbalance would not affect speedup calculation. The results are shown in Figure 5. The solid curve indicate the upper bound for speedup. In an ideal case where there is absolutely no overhead in task management and communication, the processing speed is supposed to increase linearly with the number of processors utilized for processing. The dash curve shows actual speedup. It can be seen that speedup is nearly linear when the number of processors is less than four. The performance of the system gradually degrade as the number of processors increases. The degradation is probably due

to (1) contention among processors for accessing various queues in the task queue server, and (2) overhead in inter-board communication.

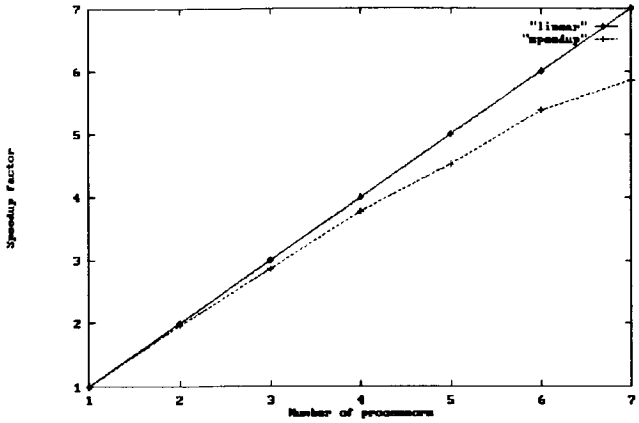


Figure 5: Speedup factors of runing parallel median filtering on MC860VS

6. Concluding Remarks

The task queue server and parallel algorithms for two vision operations have been designed and implemented on an i860-based MC860VS. Timing statistics has been collected to analyze the overhead for task management (including queue-access control and inter-board communication). For local operations which do not require merging operations, such as median filtering, nearly linear speedup can be obtained for a small number of processors and processor utilization (efficiency) gradually degrades as the number of processors increase. For any global operation which requires an additional operation to merge partial results, such as contour extraction, it may not be a good idea to divide the operation into a large number of subtasks.

Based on experimental results from initial implementation, it is expected that the proposed vision architecture will provide a convenient mechanism for composing efficient parallel and distributed programs (vision programs in particular). However, it should be pointed out that a good architecture is necessary but not sufficient for achieving real-time vision processing. For example, the first target application of the vision architecture is vision guided grasping of free flying objects in space by the ExtraVehicular Activity Helper and Retriever (EVAHR) [1]. In order to assist real-time grasping, EVAHR's vision module is required to provide poses (i.e. the orientations and locations) of to-be-grasped-objects to its arm/hand controller at 10 Hz (i.e. 0.1 second per pose estimation) [10]. Median filtering and contour extraction are part of preprocessing before pose estimation can be performed. Experimental results seem to indicate that it may take more than 0.14 seconds to perform only median filtering with 7 processors. This problem can be alleviated by applying median filtering only to subimages from which accurate information needs to be extracted. In other words, real-time vision processing cannot be achieved without efficient (sequential and parallel) vision algorithms and a good parallel vision architecture (along with powerful parallel processors).

References

[1] Grimm, K., Erickson, J., Anderson, G., Chien, C. H., Hewgill, L., Littlefield, M. and Norsworthy, N. "An experiment in vision-based autonomous grasping within a reduced gravity environment," in *Proc. SPIE Conf. on Cooperative Intelligent Robotics in Space III*, Nov. 1992, Boston, MA.

[2] Hypercube, Intel Corportation.

[3] *Training guide for the MC860 Attached Processor*, Mercury Computing Systems.

[4] Weems, Charles "The DARPA image understanding benchmark for parallel computers," Tech. Report 90-98, Dept. of Computer and Infor. Science, Univ. of Massachusetts, 1990.

[5] Bershad, B. N. and Zekauskas, M, "Midway: Shared memory parallel programming with entry consistency for distributed memory multiprocessors," Tech. Report CMU-CS-91-170, School of Computer Science, Carnegie Mellon Univ., 1991.

[6] Chien, C. H, and Lin, L. J. "PARADIGM: an architecture for distributed vision processing," in *Proc. of IEEE Conference on Pattern Recognition*, June 16-21, Atlantic City, NJ., pp. 648-653.

[7] Arnould, E., Bitz, F, Cooper, E., Kung, H. T., Samson, R., and Steenkiste, P. "The design of Nectar: A network backplane for heterogeneous multicomputers," TR-CMU-CS-89-101, School of Computer Science, Carnegie Mellon, Jan. 1989.

[8] Shafer, S., Stentz, A. and Thorpe, C. "An architecture for sensor fusion in a mobile robot," Tech. Report CMU-RI-TR-86-9, Robotics Institute, Carnegie Mellon Univ., 1986.

[9] Thorpe, C., Hebert, M., Kanade, T., and Shafer, S. "Vision and navigation for the Carnegie-Mellon Navlab, *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 10, 3, May 1988.

[10] Chien, C. H "Multi-view based pose estimation from range images," in *Proc. SPIE Conference on Cooperative Intelligent Robotics in Space III*, November 1992, Boston, MA.

MOTION ESTIMATION OF OBJECTS IN KC135 MICROGRAVITY

Lisa Hewgill
Lockheed Engineering and Sciences Company
Houston, Texas

Abstract

The ability of a autonomous space robot to grasp a freely translating and rotating object is being tested in the simulated microgravity environment aboard a KC135 airplane. The Extravehicular Activity Helper/Retriever's (EVAHR's) arm trajectory planner continually requires a current estimate of the target's translational and rotational state. The target's attitude, angular velocity and angular acceleration define its rotational state and the target's translational state include its position, velocity and acceleration. Estimators have been developed based on the extended Kalman filter (EKF) algorithm. The KC135 microgravity environment does not have a convenient inertial reference frame for the translational dynamics and therefore, the translational as well as the rotational object dynamics are described by nonlinear equations. The estimator algorithms require intensive mathematical computation and therefore, I860 microprocessors are used so that the software will run in real time. Estimator design, implementation concerns and issues specific to the KC135 environment are discussed. Translational state estimator performance results from simulation testing and from real-time integrated system testing are presented.

KC135 Experiment

One of the objectives of the Automation and Robotics Department at the Johnson Space Center is to design and develop a free-flying autonomous space robot. The immediate goal of the EVAHR project is to have the EVAHR grasp a freely translating and rotating object in a microgravity environment. This environment is simulated in the cabin of a KC135 airplane by having the plane fly along a parabolic trajectory. Sections of the robot that are necessary for the experiment are bolted to the KC135 cabin floor or are secured in some other fashion. Included are a Robotics Research arm, an inertial measurement unit (IMU), a vision system, release mechanism, cages containing the I860 processors and other computer equipment. There are two candidate vision systems: PRISM3 (1) and the Perceptron scanner (2). The PRISM3 position measurement rate is 20-30 Hz and the position measurement rate of Perceptron scanner is 5 - 8 Hz.

KC135 Environment

The gravitational forces experienced in the cabin of the KC135 change from above 1.5 g (1 g = 32.2 ft/s^2) during "pull-up" to less than 50 mg of microgravity in approximately 7 seconds. The microgravity environment (less than 100 mg) lasts approximately 20 seconds; however, the microgravity period during which the target is relatively stationary in the Robotics Research arm's work space usually ranges from 0 to 10 seconds per parabola. This decrease is caused by the fluctuations of up to 100 mg in the gravitational acceleration during the microgravity portion of the parabola. These undesired microgravity fluctuations cause the relatively stationary floating target to fly against the ceiling or floor of the airplane.

Another characteristic of the KC135 environment is the usual initial fluctuation in the vertical acceleration as the airplane enters the microgravity portion of the flight. This is illustrated in Fig. 1. To avoid premature target release and the resulting undesired target dynamics, a release indicator mechanism was developed. The vertical acceleration, the pitch rate, and the pitch acceleration are monitored to determine the proper time to release the target. These measurements are taken from accelerometers and 3-axis gyroscopes. The microgravity portions of the flight that experience minimal pitch acceleration tend to be the better quality parabolas.

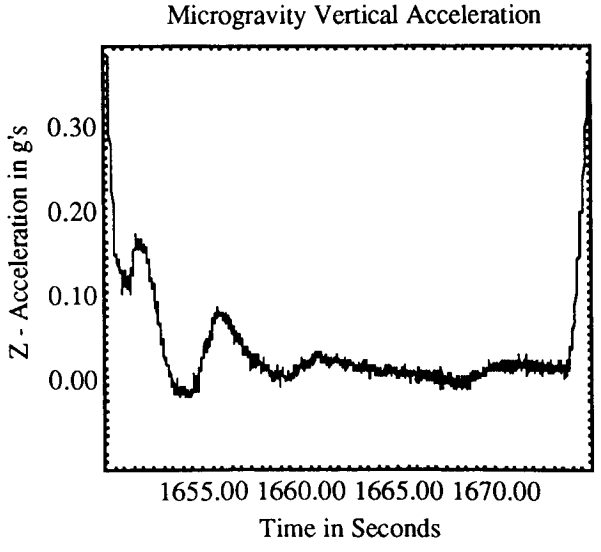


Fig. 1

Copyright c 1993 by the American Institute of Aeronautics and Astronautics, Inc. No copy is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all rights under the copyright claimed herein for government purposes. All other rights are reserved by the copyright owner.

Relative Translational Dynamics

The KC135 cabin environment and the EVAHR instrumentation do not provide a practical inertial reference frame which can be utilized in modelling the target's translational dynamics. The EVAHR translational estimator coordinate frame, which is a Cartesian coordinate system centered at a corner of the IMU plate, is essentially an accelerating reference frame. At present, the EVAHR hardware does not include a sensor that gives position expressed in an earth reference frame (differential GPS etc.). Double integrating the output of the accelerometers for 1 - 2 hours without another correcting measurement is undesirable. EVAHR instrumentation does provide enough information to calculate the target 's acceleration relative to the robot. There are several implicit assumptions in determining the relative acceleration. One is that the EVAHR's and target's gravitational accelerations are equal in magnitude. It is also assumed that the only force acting on target is gravity and third, the atmospheric drag acting on the target is zero. The relative acceleration of the target with respect to the EVAHR is given by

$$ {}^{E}a_T = -{}^{E}\omega_E \times {}^{E}\omega_E \times {}^{E}r_T - {}^{E}\alpha_E \times {}^{E}r_T - 2 * {}^{E}\omega_E \times {}^{E}v_T - {}^{E}a_E \qquad (1) $$

where

- ${}^{E}\omega_E$ is the EVAHR's angular velocity (filtered gyro reading)
- ${}^{E}\alpha_E$ is the EVAHR's angular acceleration
- ${}^{E}a_E$ is the EVAHR's translational acceleration (filtered accelerometer reading)
- ${}^{E}r_T$ is the target's position relative to EVAHR
- ${}^{E}v_T$ is the target's velocity relative to EVAHR
- ${}^{E}a_T$ is the target's acceleration relative to EVAHR.

For computational reasons, during each calculation time period (iteration) the acceleration is determined twice. The first calculation uses the target's position and velocity estimates from the previous iteration. Next, the position and velocity estimates are updated using the new relative acceleration. The acceleration is then again determined using the updated position and velocity estimates. The relative velocity and relative position are determined in each iteration as follows

(i) compute the relative acceleration, ${}^{E}a_T$, based on the target's velocity and position estimates of the previous iteration

(ii) compute

$$ {}^{E}v_T = {}^{E}vpast_T + \frac{\Delta t}{2}({}^{E}a_T + {}^{E}apast_T) \qquad (2) $$

$$ {}^{E}r_T = {}^{E}rpast_T + \frac{\Delta t}{2}({}^{E}v_T + {}^{E}vpast_T) \qquad (3) $$

where

- ${}^{E}apast_T$ is the target's relative acceleration during the last iteration
- ${}^{E}vpast_T$ is the target's relative velocity during the last iteration
- ${}^{E}rpast_T$ is the target's relative position during the last iteration
- Δt is length of iteration interval.

(iii) recompute ${}^{E}a_T$ based on updated ${}^{E}v_T$ and ${}^{E}r_T$

(iv) recompute (ii) based on updated ${}^{E}a_T$.

KC135 Translational EKF Filter Design

The EKF system model for the KC135 translational state estimator may be expressed as

$$ \frac{dx}{dt} = f(x(t)) + w(t) \qquad (4) $$

and the measurement model may be expressed as

$$ z_k = H\, x(t_k) + v_k \qquad (5) $$

where

- $x(t)$ is the state vector
- $w(t)$ is a zero mean white process
- v_k is a zero mean white sequence.

For the EKF algorithm, the state may be written as

$$ x(t) = x^*(t) + \Delta x(t) \qquad (6) $$

where

- $x^*(t)$ is an estimate of the state.

Combining equations (4) and (6), and expanding to the first order,

$$ \frac{dx^*(t)}{dt} + \frac{d\Delta x(t)}{dt} = f(x^*(t)) + \frac{\partial f(x(t))}{\partial x}\Big|_{x(t) = x^*(t)} \Delta x(t) + w(t) \qquad (7) $$

Therefore, the linearized model maybe expressed as

$$ \frac{d\Delta x(t)}{dt} = \frac{\partial f(x(t))}{\partial x}\Big|_{x(t) = x^*(t)} \Delta x(t) + w(t) \qquad (8) $$

The state matrix calculation is determined by

$$\Phi(t_{k+1}, t_k) = I + F\Delta t + (\frac{F}{dt} + F^2)\frac{\Delta t^2}{2} \quad (9)$$

where

$$F = \frac{\partial f(x(t))}{\partial x}|_{x(t_k) = x^*(t_k)}$$

Δt is the time interval between t_k and t_{k+1}

This calculation is extended out to the second order. Accuracy requirements had to be balanced against computer computation time requirements in this determination. The Kalman filter process noise covariance matrix, $Q(t_{k+1})$, is computed according to

$$Q(t_{k+1}) = \Phi(t_{k+1}, t_k)\, H\, \sigma\, H^T \Phi^T(t_{k+1}, t_k) + Q(t_k) \quad (10)$$

where

σ is a sparse matrix whose nonzero elements are associated with the variances of the white Gaussian noise of equation (20).

The Kalman filter error covariance matrix is propagated according to

$$P(t_{k+1}) = \Phi(t_{k+1}, t_k)\, P(t_k)\, \Phi^T(t_{k+1}, t_k) + Q(t_{k+1}) \quad (11)$$

The state, $x(t_k+)$, is revised by the filter according to

$$x(t_k+) = x(t_k-) + K(t_k)\,[z(t_k) - Hx(t_k-)\,] \quad (12)$$

and the Kalman gain is given by

$$K(t_k) = P(t_k-)H^T[HP(t_k-)H^T + R]^{-1} \quad (13)$$

where

R is the position measurement error covariance matrix.

Also, the Kalman filter error covariance is updated by

$$P(t_k+) = [I - K(t_k)H]P(t_k-) \quad (14)$$

The state vector specific to the relative translational state estimator is defined as follows:

$$x = \begin{bmatrix} p - 3 \text{ target position components} \\ v - 3 \text{ target velocity components} \\ \omega_e - 3 \text{ EVAHR gyro errors} \\ \alpha_e - 3 \text{ EVAHR angular acc. errors} \\ a_{2e} - 3 \text{ EVAHR accelerometer errors} \end{bmatrix} \quad (15)$$

The IMU readings and the target's position and velocity estimates, all of which have some error, must be used to determine the target's acceleration, which is given by equation (1). Now the measured angular velocity and the EVAHR angular acceleration can be written as

$$\omega = \omega_t + \delta\omega \quad (16)$$

$$\alpha = \alpha_t + \delta\alpha \quad (17)$$

where

ω is the vector of filtered EVAHR gyro readings

ω_t is the vector of true EVAHR angular velocities

$\delta\omega$ is the gyro error vector.

α is the vector of computed EVAHR angular accelerations

α_t is the vector of true EVAHR angular accelerations

$\delta\alpha$ is the angular acceleration error vector.

Similarly, the EVAHR acceleration may be written as

$$a_2 = a_{2t} + \delta a_2 \quad (18)$$

where

a_2 is the vector of filtered EVAHR accelerometer readings

a_{2t} is the vector of true EVAHR accelerations

δa_2 is the accelerometer error vector.

To form the state matrix, the time derivative of the target's velocity error is expressed as

$$\frac{d\delta v}{dt} = A\delta p + B\delta v + C\delta\omega + D\delta\alpha - \delta a_2 \quad (19)$$

where

$$A = \begin{bmatrix} \omega_1^2+\omega_2^2 & -\omega_1\omega_0+\alpha_2 & -\omega_2\omega_0-\alpha_1 \\ -\omega_0\omega_1-\alpha_2 & \omega_2^2+\omega_0^2 & -\omega_2\omega_1+\alpha_2 \\ -\omega_0\omega_2+\alpha_1 & -\omega_1\omega_2-\alpha_0 & \omega_0^2+\omega_1^2 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 2\omega_2 & -2\omega_1 \\ -2\omega_2 & 0 & 2\omega_0 \\ 2\omega_1 & -2\omega_0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

$a_{11} = -\omega_1 p_1 - \omega_2 p_2$
$a_{21} = -\omega_0 p_2 + 2\omega_1 p_0 - 2v_2$
$a_{31} = 2\omega_2 p_0 - \omega_0 p_2 + 2v_1$
$a_{12} = 2\omega_0 p_1 - \omega_1 p_0 + 2v_2$
$a_{22} = -\omega_2 p_2 - \omega_0 p_0$
$a_{32} = -\omega_1 p_2 + 2\omega_2 p_1 - 2v_0$
$a_{13} = 2\omega_0 p_2 - \omega_2 p_0 - 2v_1$
$a_{23} = 2\omega_1 p_2 - \omega_2 p_1 + 2v_0$
$a_{33} = -\omega_0 p_0 - \omega_1 p_1$

$$D = \begin{bmatrix} 0 & -p_2 & p_1 \\ p_2 & 0 & -p_0 \\ -p_1 & p_0 & 0 \end{bmatrix}$$

Therefore, the particular EKF system model is given by

$$\frac{d}{dt}\begin{bmatrix} \delta p \\ \delta v \\ \delta \omega \\ \delta \alpha \\ \delta a_2 \end{bmatrix} = \begin{bmatrix} 0 & I & 0 & 0 & 0 \\ A & B & C & D & -I \\ 0 & 0 & -\beta_1 & 0 & 0 \\ 0 & 0 & 0 & -\beta_1 & 0 \\ 0 & 0 & 0 & 0 & -\beta_2 \end{bmatrix}\begin{bmatrix} \delta p \\ \delta v \\ \delta \omega \\ \delta \alpha \\ \delta a_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ n_1 \\ n_1 \\ n_2 \end{bmatrix} \quad (20)$$

where
p is the estimate of the target's position relative to the EVAHR
$\delta p = p_{true} - p$
v is the estimate of the target's velocity relative to the EVAHR
$\delta v = v_{true} - v$
β_1 a 3x3 diagonal matrix with the value β_1
β_2 is a 3x3 diagonal matrix with the value β_2
n_1 is the white Gaussian noise vector whose value is determined from gyro noise tests and from system testing
n_2 is the white Gaussian noise vector whose value is determined from accelerometer noise tests and from system testing

Equation (20) has the same form as equation (8).

Rotational State Estimator

The rotational state estimator uses quaternions to describe the target's attitude and utilizes an EKF algorithm [3]. For further information on quaternion estimation refer to Bar-Itzhack's work [4]. Bierman's U-D factorization [5] technique was implemented to test for improved performance. An initial difference in performance was noted between the conventional EKF and the U-D factorization algorithm. However, after convergence, there was not a marked difference in performance.

The essential difference between the rotational state estimator intended for the earth orbit scenario and the KC135 rotational state estimator is the definition of the inertial reference frame. For the KC135, the inertial reference frame for the target's attitude will be the EVAHR's coordinate frame when the microgravity portion of each parabola commences (t=0). The EVAHR's attitude will be determined by integrating the output of the gyros from the commencement time (t=0).

Results

The performance results shown in Fig. 2 are the product of a simulation test. A program was developed that simulates the dynamics of KC135 airplane, the IMU sensor readings, and the dynamics of the target. The added noise is white Gaussian noise. The Euclidian norm error of the position vector estimate and of the position vector measurement are shown.

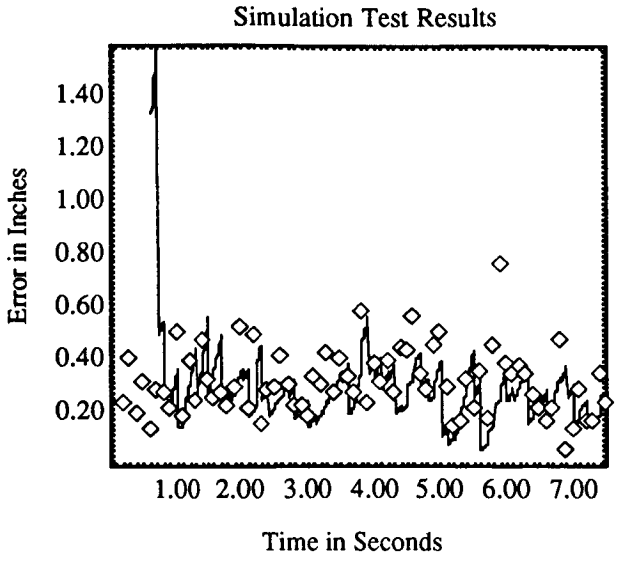


Fig. 2

Testing of the Perceptron scanner is ongoing and calibration of PRISM3 vision system is in process at the writing of this paper. Results shown in Fig. 3 are preliminary. For this test the target was stationary and Fig. 3 shows the target's x-position estimate as well as the x-position measurement. Once calibration is complete, the values of measurement error covariance matrix **R** of equation (14) will be modified.

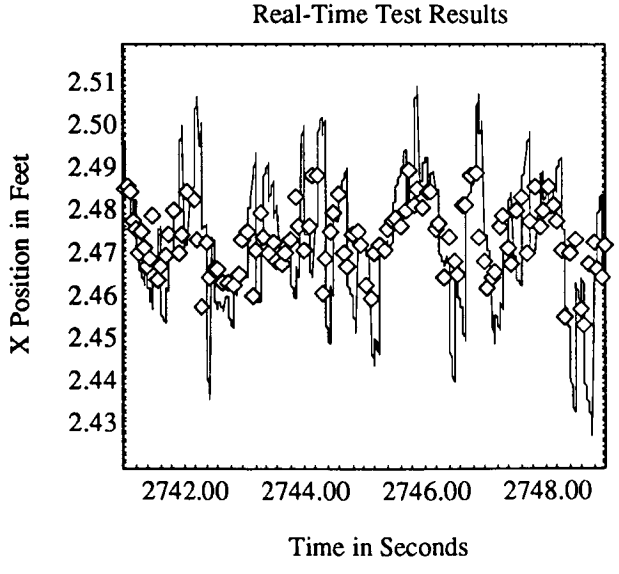


Fig. 3

Implementation Issues

For the arm trajectory planner software to be able to receive the target's state at the rate of 100 Hz, the relative translational state estimator had to be divided into a repropagation unit and real-time propagation unit. The repropagation unit accepts dated position measurement inputs, compares the measurement to the appropriate archived state, revises the state, and then repropagates the state to present time. The updated state is passed to the real-time propagation unit. This module replaces its target state and Kalman filter parameters with the new updated state and new Kalman filter parameters. The real-time propagation unit continues to propagate the state and appropriate Kalman filter parameters while the repropagation unit has accepted another dated position measurement and is revising another archived estimate. The architecture is shown below in Fig. 4.

The estimators are implemented on Mercury I860 processors because of their fast computation capability. Closed form solutions were used where possible and matrix computation decreased by taking advantage of sparse matrices.

A sensor bias test is performed just prior to the start of the flight, and these values are used to correct the IMU readings. The expected IMU drift during the flight was determined by running drift tests for a length of time comparable to flight duration, (1 - 2 hr). The sensor outputs are also passed through hardware and software filters before the readings are fed into the estimator at the rate of 100 Hz.

Summary

The design and performance of the KC135 translational state estimator were discussed. Also presented were issues specific to the KC135 microgravity environment.

Acknowledgements

The author wishes to acknowledge the significant contributions of Mr. Greg Anderson with the development of the KC135 dynamics and of Mr. Robert Norsworthy with the implementation of the translational estimator on the I860 processors, both of whom are colleagues on the EVAHR project.

References

1. E. Huber, "Object Tracking with Stereo Vision", to be presented at the AIAA Conference on Intelligent Robots for Factory, Field, Service, and Space, Houston, Tx, Mar, 20 - 24, 1994

2. M. Littlefield, "Real-Time Tracking of Objects in a KC-135 Microgravity Experiment", to be presented at the AIAA Conference on Intelligent Robots for Factory, Field, Service, and Space, Houston, Tx, Mar, 20 - 24, 1994

3. L. Hewgill, "Motion estimation of a freely rotating body in earth orbit", SPIE Cooperative Intelligent Robotics in Space III Proceedings, Boston, MA., Nov. 16 -18, 1992. Vol. 1829, pp. 444 - 457

4. I. Y. Bar-Itzhack and Y. Oshman, "Attitude Determination from Vector Observation: Quaternion Estimation", *IEEE Transactions on Aerospace and Electronic Systems*, pp. 128 - 136, Jan. 1985

5. G. H. Bierman, "Measurement updating using the U-D factorization", IEEE Conference on Decision and Control, 6th and Symposium on Adaptive Process, 14th Proceedings, Houston, TX, Dec. 10 - 12, 1975. pp. 337 - 346, Institute of Electrical and Electronic Engineers, New York, 1975

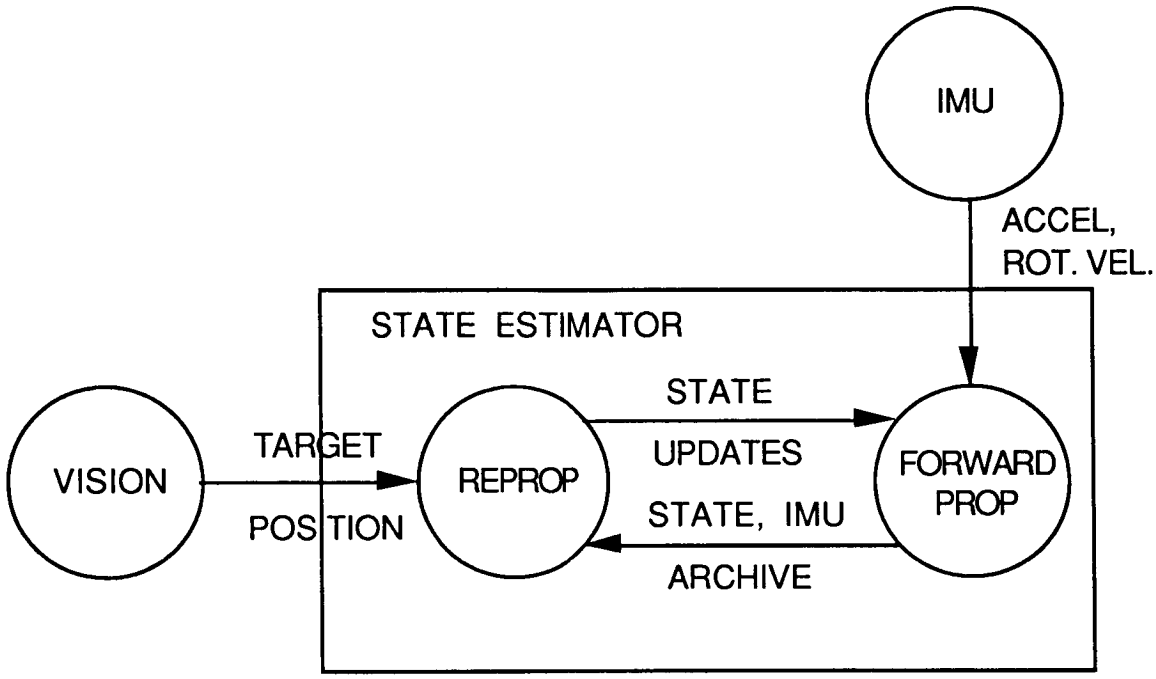


Fig. 4 Architecture of KC135 Translational State Estimator

Real-time Tracking of Objects for a KC-135 Microgravity Experiment

Mark L. Littlefield
Intelligent Systems Department
Lockheed Engineering and Sciences Company
2400 Nasa Road 1
Houston, TX 77058
mll@lobo.jsc.nasa.gov

Abstract

This paper outlines the design of a visual tracking system for use on the Extra-Vehicular Activity Helper/Retriever (EVAHR) autonomous robot during tests on board NASA's KC-135 Reduced Gravity Laboratory. Issues such as the laboratory environment, mission requirements, real-time constraints, and computational loads will be examined.

EVAHR is an autonomous robot designed to perform a number of tasks in an on-orbit microgravity environment. One of the critical tasks of EVAHR is the ability to grasp a freely translating and rotating object. This task is the current focus of investigation by the EVAHR development team. To perform this task, EVAHR must analyze range image generated by the primary visual sensor to locate and focus its sensors on the target so that an accurate set of object poses can be determined and a grasp strategy planned. This may involve positioning the sensor with its gimbal (pan/tilt) system and adjusting sensor parameters to provide the perception system with the best possible views of the target. The tracker must also provide the information that it extracts about the target to pose and state estimation modules. These tasks must be performed at a frame rate of ~9 frames per second.

1. Introduction

The EVA Helper/Retriever (EVAHR) is a prototype robot currently undergoing development at NASA's Johnson Space Center. The goal of the EVAHR project is to develop an autonomous robot which can carry out on-orbit tasks such as inspection, worksite preparation, Orbital Replacable Unit (ORU) changeout, and crew and equipment retrieval (CERS)[1].

The initial task chosen by the EVAHR team for investigation is the CERS task. The goal for this task is to autonomously grasp freely floating and rotating objects. Given the situation in which an object (crew member, tool, ORU, etc.) becomes unteathered and drifts away from a work area, the EVAHR must locate, rendezvous, grapple, and return with the object in a reasonable amount of time.

To accomplish this task in a realistic manner, a system consisting of a Robotics Research k-807i arm, a dexterous manipulator, a Perceptron laser range scanner mounted on a high-speed pan/tilt, and an instrument package consisting of gyroscopes and accelerometers (an Inertial Measurement Unit, or IMU) has been assembled, and a series of flights aboard NASA's Reduced Gravity Laboratory (KC-135 aircraft) has been scheduled[2]. Software for this task consists of an object tracker, a pose estimator, a set of state estimators (translational and rotational) and an arm/hand control module[3].

To prepare for this task, the software modules were tested and debugged against an on-orbit simulator[4] for testing the EVAHR software over an entire rendezvous and grasp. The simulator models the environment around a space station and maintains dynamics on the EVAHR and any other free-floating objects. The simulator also generates range images of the environment with the same general characteristics as the Perceptron laser range scanner. This simulation also has the ability to remove some modules and substitute perfect data to others, which provides an excellent testbed for debugging modules and verifying algorithms.

The second phase of investigation consists of a series of experiments aboard a KC-135 parabolic aircraft. There are four sets of flights scheduled. The first flight set consisted of a series of flights using the IMU to measure and record the aircraft motion during the parabolas. This information allowed the EVAHR team to characterize the aircraft motions and design the state estimators and the target release indicator needed for the following flights.

For the second flight, a simplified tracker program was flown with the laser scanner and pan/tilt device to evaluate the hardware performance and to collect data on object motion during the flight. The robot arm and hand were also flown but each ran a series of independent tests to ver-

Copyright ©1993 by the American Institute of Aeronautics and Astronautics, Inc. No copyright is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all rights under the copyright clamed herin for Governmental purposes. All other rights are reserved by the copyright owner.

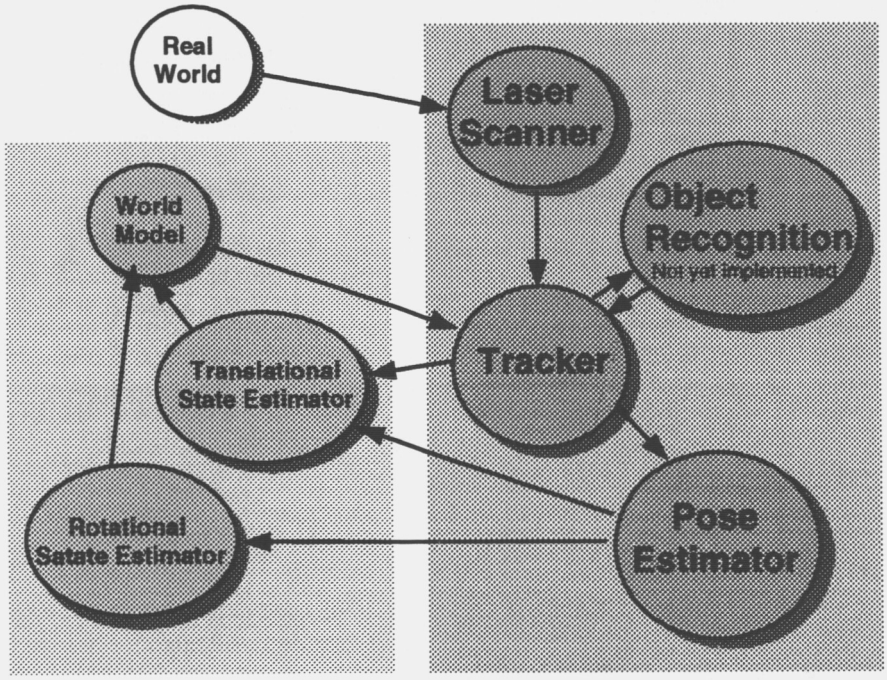


Figure 1: EVAHR perception modules and data flow

ify the hardware and software control module performance.

The third flight consists of flying the entire hardware configuration in a closed loop with the tracker, translational state estimator, and arm control modules, to grasp a ball during micro-g. Finally, in the fourth flight, the system will grasp a rotating polygonal target with the stage three system, combined with the pose estimator and rotational state estimator modules.

Phase I and flights 1 and 2 of phase II have been completed. Flight 3 is currently scheduled for February 1994 and flight 4 is scheduled for late Q1 1994[3].

The primary sensor for the EVAHR is a Perceptron laser range scanner mounted on a pan-tilt device. This scanner provides 12 bit range images over a 15 meter range. The scanner generates an image by sweeping a modulated infrared laser across the field of view using two mirrors, a spinning mirror (for each scan line) and a panning mirror (to move the scan lines up and down). The depth at each pixel is measured by comparing the phase of the returning laser signal.

The scanner has three speeds of operation (the slower the speed, the higher the spatial resolution) and a variable vertical field of view. At it's highest speed, the scanner produces 2.25 frames per second at 256x256 pixels, or 9 frames per second at 64x256 pixels. The latter is the setting sued in most of the EVAHR experiments. The scanner also provides a pixel registered 12 bit intensity image along with the range image.

There are three computational components that currently make up the perception system (figure 1). First, an object tracker performs a series of image processing tasks, and identifies the objects in the range images from frame to frame. The second module is the pose estimator[5], which calculates the object location and orientation (pose). The third module consists of the object state estimators[6], which maintain a real-time state of the object's rotation and translation. These three modules form a cross connected loop, in which each module provides information to the other modules as information becomes available.

This paper describes the tracker module of the EVAHR. Section 2 describes general design of the tracker module and the results of the initial test series in the on-orbit simulation. Section 3 describes the stage II/flight 2 tracker and the results of that flight. Section 4 describes the stage II/flight 3 tracker, focusing mainly on the lessons learned from flight 1. Section 5 discusses the added features required for the stage II/flight 4 tests and section 6 is a summary and a discussion of the future plans for the vision loop and the tracker module in particular.

2. System design and on-orbit simulations

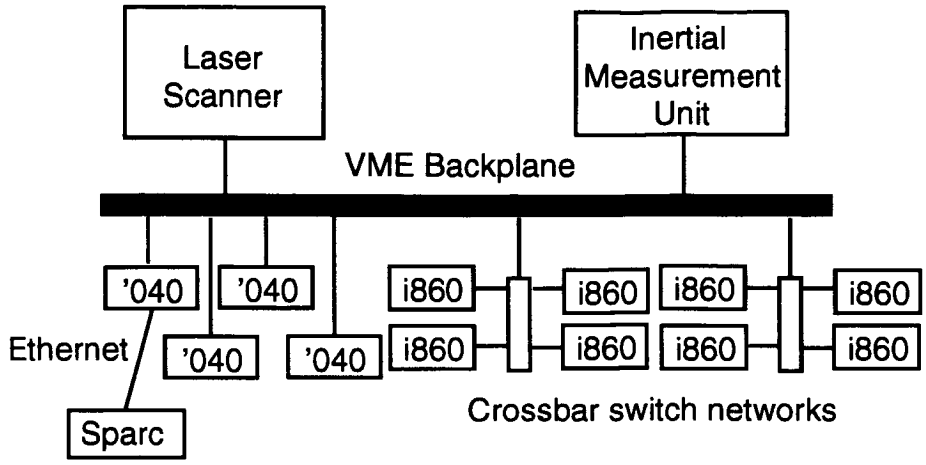


Figure 2: Physical connections for EVAHR percpetion hardware.

NASA's Reduced Gravity Laboratory is a KC-135 aircraft which is flown in a parabolic path to produce up to 25 seconds of freefall. Of this 25 seconds, roughly 9-15 seconds have less than 30 milli-g of Z axis accelerations. This is the period in which EVAHR must perform it's grasp.

To accomplish this task, the tracker module was designed as the first stage in the perception system. In the preliminary design, the tracker module had five tasks[7]:

1. Locate the target in each image. This also involves locating and marking the arm in the image.
2. Send the target position to the translational state estimator.
3. Send target contour data to the pose estimator.
4. Maintain an internal object database on target parameters (position, velocity, confidence, etc.).
5. Calculate the scanner configuration values and gimbal positions to maintain a lock on the target.

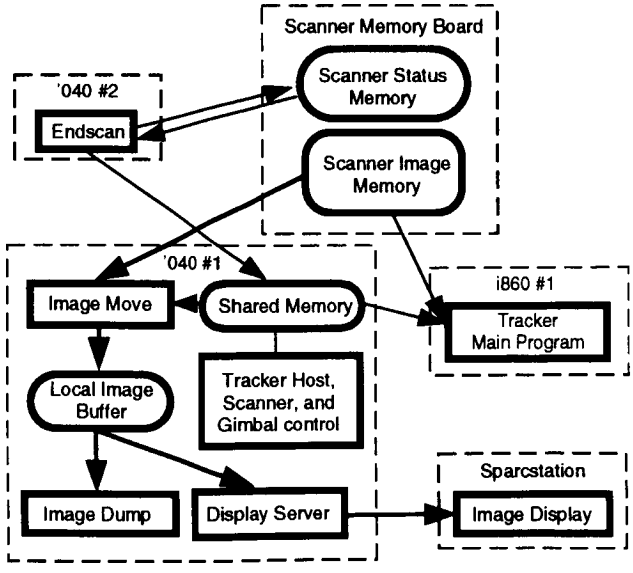


Figure 3: Sub-program organization and data flow in the tracker module.

The code to perform these tasks was tested extensively on an on-orbit simulator. Although the simulator provided an environment in which techniques could be developed and data structures could be ironed out, there were several shortcomings. First, the data provided from the scanner simulator was noise free. This made object segmentation trivial as pixels were either object, EVAHR arm, or background. Also, control of the scanner parameters and gimbal positions was instantaneous and took nothing more than a message to the scanner simulator. Finally, the scanner simulator did not perfectly simulate the Perceptron scanner, neither in the geometry of the image that it generated, nor in the way in which it could be configured.

For the flight test tracker module, several additional tasks were added:

6. Command and control of the scanner and gimbal.
7. Maintain a frame synchronization on the Perceptron scanner.
8. Optionally display or store images periodically for operator feedback or off-line analysis.

Each of these tasks, plus the ones listed above, must be performed in the 0.11 second cycle provided by the generation of the images.

To accomplish this, the tracker module is split into several concurrent sub-programs running on four separate computers: two 68040-based, vxWorks machines and a Mercury i860-based machine mounted on a VME backplane, and a Sun Sparcstation connected to one of the vxWorks machines via ethernet (figures 2 and 3). A host program, running on an '040 spawns the other programs, synchronizes them, and commands the scanner and gimbal. Also on this '040 are the image move, image display server, and image dump sub-programs. The image move sub-program

moves images from scanner memory into a local buffer on the '040. The image display server sub-program connects to the image display sub-program running on the Sparcstation via ethernet and sends the locally stored images as fast as the bandwidth of the ethernet allows. The image dump sub-program takes locally stored image and stores them on a backplane-mounted hard disk.

The second '040 machine runs a sub-program (endscan) which performs two tasks: reset the scanner at the end of each image and maintain a data structure on the currently available image which records the address of the image in scanner memory and the direction of the scan. To maximize the speed that it collects images, the Perceptron scanner collects images on both the down scan and up scan of the mirror (there is no vertical retrace). This means that while the even images are collected top to bottom, the odd images are collected bottom to top, meaning that they will be "upside down" in memory. It is up to the controlling program to keep track of the direction of the scan for each image collected (there is no indicator of scan direction).

To collect data, the scanner must be commanded to 1) start the panning mirror to begin panning and 2) start collecting data[8]. This is done by setting a STOP ADDRESS for data collection, sending a "B" (for bi-directional scanning) over the serial port, and setting a FRAME REQUEST flag in the scanner status register. When the scanner begins storing data it asserts a FRAME BUSY bit in the status register and continues to assert it until the commanded number of pixels worth of data have been collected. The scanner also records the address of the current 4K block that it is writing to in the status register. At the end of the scan the scanner de-asserts the FRAME BUSY bit, but does not stop the panning mirror's motion. The endscan sub-program must poll the FRAME BUSY bit, and when it is de-asserted, set the STOP ADDRESS for the next frame and set the FRAME REQUEST flag before the mirror has reversed it's direction and started motion again. It typically takes between 2.15 and 10.77 milliseconds to perform this task. If endscan fails to perform this task, the FRAME REQUEST will not be recognized by the scanner and the upscan/downscan synchronization will be lost.

The main tracker sub-program runs on a Mercury i860. There were two reasons for the placing it on this platform. The first was the raw speed of the i860-based machine. The second was to facilitate communications between the tracker and the state and pose estimation modules. On the Mercury system, four i860 are connected by a crossbar switch, allowing shared memory access at internal memory access speeds.

The first task of the main tracker sub-program is to segment and identify each object blob found in each image from the laser range scanner. A unique object ID is given to each object that is found. As objects move relative to the scanner, their blobs move in the image frame. The tracker must maintain a proper object ID on each object in the field of view for each frame from the laser scanner, and send this information, along with 3D blob contour data, to the pose estimator. No object recognition is performed at this stage. To simplify the problem for the KC-135 experiments, it is assumed that a single object will be visible at the start of micro-g and that it is the target object. If, by chance, additional objects appear during micro-g, the tracker utilizes predicted location and object size to correspond the target object.

3. Stage II/Flight 2: hardware validation and data collection

The second flight in the planned EVAHR KC-135 experiments is to test the hardware on board the KC-135 during a flight to validate it's performance and to collect data on object motions during the micro-g potions of the flight. A simplified version of the tracker program tested in the on-orbit simulator was used to try and keep the scanner pointed and focused on the target (white ball) and to store images for off-line analysis. A micro-g indicator was also flown to inform the operator when to release the target.

During this flight, a crewmember released the target in front of the scanner at the start of micro-g. The simplified tracker then segmented any discrete blobs in each of the range images generated by the scanner, picked the blob that was "most circular", and made sure that the mapper was focused and pointed at the target that it located in the image. Meanwhile, the image dump sub-program saved images to the local hard disk. No information about the object was saved or used in subsequent images.

During preliminary tests aboard the KC-135, it was discovered that the first few seconds of micro-g are very noisy. That is, there are relatively large accelerations experienced during this time. To detect when this period is over and "clean" micro-g occurs, a release indicator was developed. The release indicator analyzes the data from the IMU and activates a light when the micro-g has stablized[9].

During some of the parabolas, the crewmember released the target when signalled by the release indicator while during others the crewmember used their own senses to estimate the proper time to release the target. In general, if the crewmember released the target when signalled by the release indicator rather than guessing on their own, there was less unwanted motion in the target object.

It was discovered during early testing that moving the gimbal while the scanner was scanning an image would inter-

rupt the scan and disrupt the upscan/downscan synchronization. This is due to a safety device which shuts down the laser if the spinning mirror speed is not within a specified tolerance. To overcome this problem the tracker waits for a scan to complete, commands the scanner to stop scanning, moves the gimbal, then commands the scanner to start scanning again. This is a rather time consuming task, taking on the order of 0.75-1.0 seconds to complete.

To keep from gimballing unnecessarily, the tracker makes use of a feature of the scanner in which the vertical field of view can be adjusted to start and stop anywhere in the 60 degree overall vertical field of view. This means that when the scanner is set to focus on a target with, say, a 15 degree vertical field of view, and the target moves toward the top or bottom of the image, the scanner can be commanded to adjust the field of view up or down, thus maintaining the target in the center of the image. The tracker only moves the gimbal when the top or bottom of the vertical field of view moves outside the 60 degree overall field of view, or if the target moves close to the left or right sides of the image (a 60 degree field of view at the motor speeds we use during the flight).

During the off-line analysis, it was discovered that the tracker actually failed to track during most parabolas. There were two reasons for this:

1. Segmentation failed when the target moved too close to the walls or floor of the aircraft.
2. The target moved out of the field of view during scanner reconfiguration or gimbal motion.

Although data was successfully collected during the parabolas that the tracker functioned correctly, nearly all data was lost for the rest. For the following test flights, significant changes were made.

Several lessons were learned from flight 1:

1. Target segmentation using only the range image was inadequate.
2. Control of the scanner and gimbal was too slow.
3. Focus of the field of view was too tight.
4. There was no search mechanism for lost targets.
5. Having a crewmember inside the field of view complicated both the segmentation and target location.
6. Waiting until the release indicator signals to release the target reduces the amount of unwanted target motion dramatically.

4. Flight 3: Ball grasp

Flight 3 is the first flight where a grasp of an object is attempted. For this flight, it is imperative that the tracker maintain a lock on the target throughout the micro-g portions of the parabola.

Using what was learned in flight 2, changes were made to the tracker software. The elements of the on-orbit simulation version of the tracker that were left out for flight 2 were added in. These additions maintain a database of objects seen by the tracker and attempts to maintain a correspondence between objects located in each image, and the objects in the internal database. Code was also added to located and label the arm in each image[7].

Other changes were made to the tracker to address the issues raised after flight 2. These changes included:

1. Making the target white, the background black, and segmenting on the reflectance image generated by the scanner.
2. Increasing the speed of the commands to the scanner and the gimbal.
3. Adjusting the mechanism for focusing the field of view to be more conservative.
4. Adding a mechanism for searching for the object if it becomes lost.
5. Using a release mechanism to release the target, rather than a crewmember. This was actually planned for flight 3 to protect crewmembers from entering the workspace of the arm.
6. Planning to rely on the release indicator to signal the target release.

The most significant change to the tracker module is the change from segmenting based on the range image to segmenting from the reflectance image. This requires that the interior of the aircraft be covered in black fabric and the ball to be painted white. The arm and hand must remain their original white/aluminum colors, however, so locating the arm/hand in the image still must be performed.

Another problem that plagued the tracker during flight 2 was the loss of the target while setting scanner configurations or moving the gimbal. This problem was solved using three techniques. First, the communications between the tracker and the scanner and gimbal controller was speeded up. The time now needed to change the scanner settings is ~0.07-0.10 seconds, verses ~0.3 seconds during flight 2. Also, the time for gimbal motion has been cut from ~0.75-1.0 seconds, to <0.5 seconds.

The second technique for attacking the target loss problems is to adjust the vertical field of view to it's maximum following a gimbal move. This maximizes the chance that the target will be in the field of view. Finally, when a gimbal move is needed, the scanner is moved in both pan and

tilt, pointing the scanner at the target. This is opposed to the method used in flight 2 in which the gimbal was moved only in pan if the target is at the right or left edge of the image and moving only in tilt if the target is at the top or bottom of the field of view.

During flight 2 there was no mechanism for recovering lost objects. If the target was lost, the tracker moved back to it's home position. To correct this the tracker uses a layered approach for reaquiring the target. If no target can be located in an image, the tracker commands the scanner to open the vertical field of view to it's widest. If, after the scanner change, no target can be found, the tracker commands the gimbal to point the scanner at the predicted location of the target (at the estimated time that the gimbal motion should be complete). This predicted location is generated either from the translational state estimator or the internal tracker estimate of the target position and velocity. If there is no target found after these two operations, the target is considered "lost". At this point the scanner and gimbal are reset to their starting positions and the tracker data structures are cleaned up and reset.

Although originally scheduled for Q4 1993, flight 2 has been postponed until Q1 1994 due to hardware and KC-135 scheduling problems.

5. Flight 4: Polygonal object grasp

For flight 4, the tracker software will remain generally the same, with the addition of the communications with the pose estimator module. This flight is currently scheduled for late Q1 1994.

When the pose estimator is ready for data, the tracker sends a contour of the target object, labeled with both depth information and which portions of the contour belong to the target (in the case of occlusion). Along with this information extracted from the image, the tracker passes along the state of the scanner and gimbal, and the estimated pose of the target (if it is available).

There are two competing issues that the tracker must deal with during the flight 4 object grasp:

1. The pose estimator takes a relatively long period of time to calculate the pose of an object without any prior knowledge of the target pose.
2. The rotational state estimator must have data very fast during it's initial start-up period in order to converge to a solution in a reasonable time.

To overcome these issues, the tracker queues up the first three images worth of data to pass to the pose estimator, whenever the pose estimator is free. If the rotational state estimator gets bad data from the state estimator, or if it cannot converge with the three images worth of data that it receives, it tells the tracker to re-initialize. The tracker then queues another three images worth of data and the process is repeated. This task is performed as part of the main tracker image processing cycle.

6. Summary

The tracker module performs the first stage of image processing as part of a general perception system for EVAHR. In addition to locating a target object in a series of range image, the tracker must transmit the position of the target to the translational state estimator, transmit range data from the target contour to the pose estimator, and correctly configure the scanner and point the gimbal so that the target stays inside the image. These tasks must me performed within the real-time constrains of the scanner frame rate and the environment inside NASA's Reduced Gravity Laboratory KC-135.

Of four sets of flights scheduled aboard the KC-135, two have been completed as of the writing of this paper. The first set of flights measured the motion of the aircraft with an inertial measurement unit (IMU). The second set of flights involved flying the laser scanner, the arm, and the hand in a series of independent tests. Also during these flights the tracker system stored images of a floating target to a hard disk for off-line analysis.

The third set of flights will involve using the perception system, along with the arm and hand systems, to grasp a freely floating ball. The fourth and final set of flights will incorporate a pose estimator and rotational state estimator to grasp a polygonal target.

The tracker module plays an important role in flights 2, 3, and 4. For flight 2, the tracker was tasked to maintain the scanner pointed toward a ball target throughout the micro-g portions of the flight. Although the tracker largely failed in it's task, important lessons were learned.

For flights 3 and 4, the tracker will maintain a history of each object that it detects in it's images. It uses this information to maintain an object correspondence between frames. The tracker also has a method for searching for the target if by chance the target moves outside the field of view of the scanner. This allows the tracker to recover from unexpected events such as an unexpected aircraft motion.

6. References

1. Crumrine, S.B., Dellenback, S.W., Fink, P.K., Franke, E.A., McFalls, D.S., *Requirements for an Extra Vehicular Activity (EVA) Robotic Assistant, Final Report*, Crew and Thermal Systems Division, NASA Johnson Space Center, Houston, 1987.

2. White, Linda G., *JSC Reduced Gravity Program Users Guide, JSC-22803*, Flight Crew Operations Directorate, Aircraft Operations Division, July 1991.

3. Grimm, K., Erickson, J.D., Norsworthy, R., Anderson, G., Hewgill, L., Littlefield, M., Chien, C.H., "An experiment in vision-based autonomous grasping within a reduced gravity environment," *SPIE Cooperative Intelligent Robotics in Space III*, Boston Massachusetts, 1992.

4. Norsworthy, R.S. and Merkel, L.E., *EVA Retriever and On-Orbit Environment Simulation Requirements and Design*, report number LESC-28938, Lockheed Engineering and Sciences Company, NASA Johnson Space Center, 1990.

5. Chien, C.H., "Multi-view based pose estimation from range images," *SPIE Cooperative Intelligent Robotics in Space III*, Boston Massachusetts, 1992.

6. Hewgill, L, "Motion estimation of a freely-rotating body in earth orbit," *SPIE Cooperative Intelligent Robotics in Space III*, Boston Massachusetts, 1992.

7. Littlefield, M.L., "Adaptive tracking of objects for a mobile robot using range images", *SPIE Cooperative Intelligent Robotics in Space III*, Boston Massachusetts, 1992.

8. *Perceptron LASAR Hardware Manual*, Perceptron, Inc. 1992.

9. Hewgill, L., "Motion estimation of objects in KC-135 micro-gravity", AIAA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS), Houston, Texas, March 1994.

Grasping Objects Autonomously in Simulated KC-135 Zero-G

Robert S. Norsworthy
Intelligent Systems Department
Lockheed Engineering & Sciences Co.
Houston, Tx 77058
robertn@superman.jsc.nasa.gov

ABSTRACT

The EVAHR (Extravehicular Activity Helper/Retriever) robot is being developed to perform a variety of navigation and manipulation tasks under astronaut supervision. The EVAHR is equipped with a manipulator and dexterous end-effector for capture and a laser range imager with pan/tilt for target perception. Perception software has been developed to perform target pose estimation, tracking, and motion estimation for rigid, freely rotating, polyhedral objects. Manipulator grasp planning and trajectory control software has also been developed to grasp targets while avoiding collisions.

Flight experiments have been scheduled aboard NASA's Reduced Gravity Laboratory (KC-135 aircraft) in 1994 to attempt grasps of free-floating objects. A software simulation of the EVAHR hardware, KC-135 flight dynamics, collision detection, and grasp impact dynamics has been developed to integrate and test the EVAHR software. This paper describes the EVAHR system, with emphasis on the robotic and KC-135 simulation software.

1. INTRODUCTION

Much work has been done on autonomous grasping of stationary, complex objects, e.g. in bin-picking and pick-and-place problems[1,2]. However, very little work has been devoted to autonomous grasping of moving, complex objects. This is probably due in part to the difficulty of the problem and to a lack of applications for such technology. There exists an abundance of applications for this technology at NASA, however, as recent Shuttle-based satellite recovery attempts have amply demonstrated. STS-49 required 3 astronauts to grasp the ailing INTELSAT VI satellite - an autonomous mechanism would have spared the astronauts hazardous EVA activity. The difficulties encountered on 41-C in 1984 rescuing the Solar Max satellite and the failure to recover the Leasat 3 during 51D in 1985 provide further evidence of such a need. The recent ROTEX experiment onboard the Shuttle mission STS-55[3] demonstrates that the German Aerospace Research Establishment (DLR) considers this a key piece of robotic technology for space.

Copyright © 1994 by the American Institute of Aeronautics and Astronautics, Inc. No copyright is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all rights under the copyright claimed herein for Governmental purposes. All other rights are reserved by the copyright owner.

The ROTEX system combined a tele-operated manipulator and camera system to grasp a free-floating object, among other tasks.

The EVAHR (Extra-Vehicular Activity Helper/Retriever) project has been developing robotic technology, hardware and software, for navigating among and grasping free-floating objects since 1987. The EVAHR project was originally conceived as an autonomous device for retrieving personnel or objects which had become detached from Space Station Freedom (SSF). At the conclusion of Phase II in 1990, the following autonomous robotic capabilities were demonstrated on a precision air bearing floor (PABF): a) navigation among stationary obstacles and b) grasping arbitrarily located and oriented, cylindrical targets. At this time, a simulation of the EVAHR in an 6 degree-of-freedom (DOF) orbital environment[4] was developed for use in the next phase. It would provide a testbed for development of 6 DOF navigation and grasping software. Models initially developed included orbital dynamics, plume impingement effects of the Manned Maneuvering Unit (MMU) (a compressed-gas-based propulsion system), and a laser scanner range/intensity image simulation[5]. Six DOF body motion control and orbital state estimation algorithms were developed and tested for navigation purposes. Phase III then took as its primary goal the development and demonstration of an integrated system for grasping free-floating, complex objects.

The EVAHR team chose the KC-135 zero-gravity aircraft as the hardware testing and demonstration platform for this phase[6]. The KC-135 aircraft flies a series of parabolas, free-falling over the hump of each parabola, simulating zero-gravity or weightlessness for objects and personnel free to float inside the plane. Each of these periods lasts 20-25 seconds. The EVAHR, attached to the floor of the KC-135 throughout the experiments, will attempt to grasp targets released in its envelope during the simulated zero-gravity period . While the KC-135 is a difficult environment for such testing, it provides the only direct means of testing the contact dynamics of free-space grasping short of a Shuttle flight experiment.

Prior to flight, software to grasp free-floating, complex objects in a 6 DOF, orbital environment was developed and integrated. Software for an orbital environment was developed first and testing was conducted to determine the performance of the system[7]. Dynamics and state estimation software was then developed to represent the KC-135 environment.

2. OBJECTIVES

Essentially, the integrated EVAHR grasping software must be capable of **autonomously grasping a variety of polyhedral objects, each in less than 15 seconds, where the object is freely but slowly translating and freely rotating at up to 30 deg/sec**.

A maximum time-to-grasp of 15 seconds was initially chosen as a conservative estimate of the zero-gravity period provided by the KC-135. While recent data gathering flights on-board the KC-135 have shown that the time available is actually less than 8 seconds, most of the testing presented here was conducted with the 15 second per grasp attempt time limit. One might speculate that in order to be considered useful in real applications, the robot would have to perform these grasps much more quickly, but this consideration did not affect our requirements.

The maximum rotational rate is taken from a Crew and Equipment Retrieval Study (CERS) conducted by NASA[8], which estimated the likely maximum separation rates of objects which might become detached or untethered from the SSF. The CERS maximum translational rate of 2 feet/sec was ignored because the EVAHR, as a free-flying vehicle, would rendezvous with and stationkeep at the target with nearly the same velocity. From on-orbit navigation tests in the EVAHR simulation, a rate of 2 in/sec during stationkeeping was found to be the maximum. However, it would be impractical for the EVAHR vehicle to match the CERS rotational rates.

To enable the EVAHR to meet these objectives, a metrically-accurate CAD surface model and mass properties (mass, center-of-mass, inertia matrix) are provided to the EVAHR describing each target object prior to runtime. The target's surface must be non-specular so that the laser scanner can image it.

Integrated software testing with the orbital simulation yielded information about the performance of the EVAHR software in its ultimate target environment, i.e. the success ratio for different target velocities, the required vision update rates, etc. The addition of a KC-135 dynamics model provided a means of integrating the KC-135 state estimation module and verifying continued, successful system performance.

3. EVAHR HARDWARE

The principal EVAHR hardware elements to be flown on the KC-135 flight experiments is shown in Figures 1 and 2. These elements are combined into a fictitious EVAHR body in the orbital simulation shown in Figure 3. A Perceptron LASAR laser scanner is mounted in a pan/tilt mechanism on top of the EVAHR body. The Perceptron provides 64x256 range/reflectance images at 9 Hz, 128x256 resolution at 5 Hz, or 256x256 at 2.5 Hz - switchable at runtime. The range resolution is approximately 1/7 inch, though noise can result in a range error of up to 3%. The pan/tilt motors are able to rotate the Perceptron at 180 deg/sec. A Robotics Research (RR) K807i 7 DOF manipulator is mounted as a left arm, with a 3-fingered JH4 dexterous hand as its end-effector. The RR arm has a maximum end-effector speed of 30 in/sec, static repeatability of .002 in, and may accurately support loads of 10 lbs. The dexterous hand was manufactured at JSC. It provides 3 joints per finger, though only 2 are active. The maximum finger joint speed is roughly 120 deg/sec so that the hand is able to close in roughly 1/4 sec. Proximity detectors are located on the palm and each finger tip.

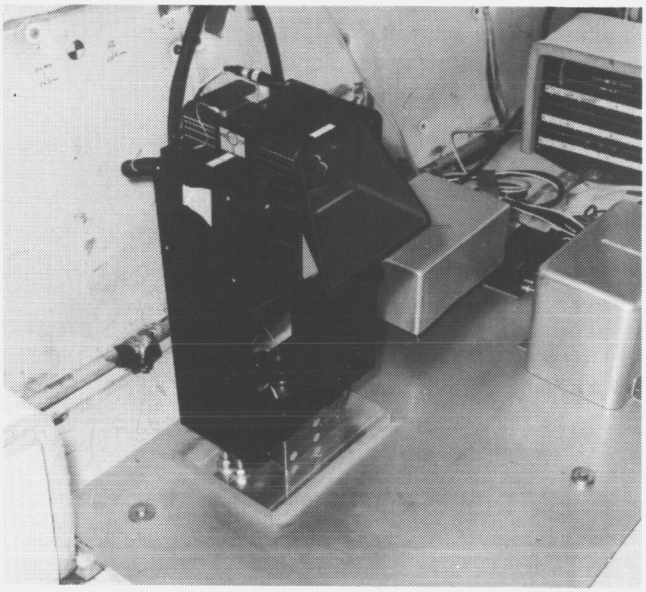


Figure 1. EVAHR laser scanner & pan/tilt

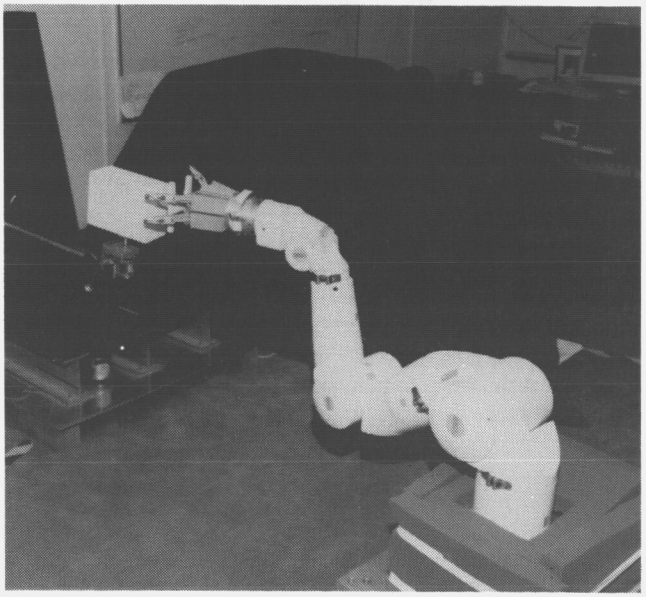
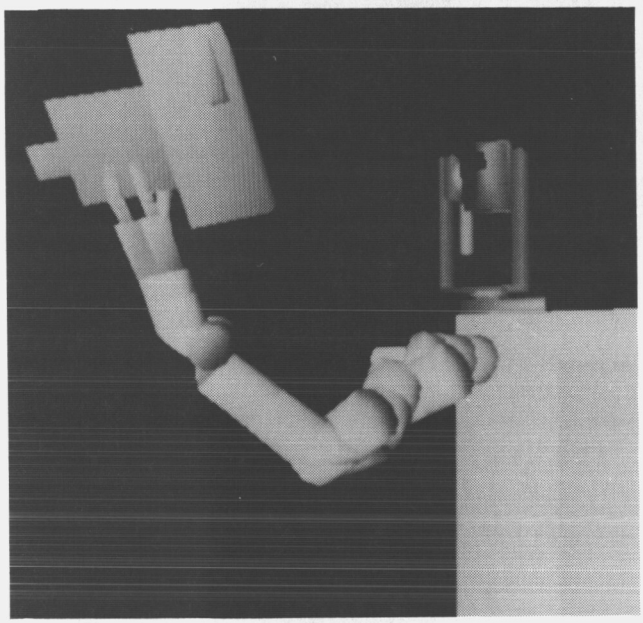


Figure 2. EVAHR manipulator/hand

When the EVAHR is flown on the KC-135, the EVAHR hardware will be fixed to the floor of the aircraft - only the target will float free. In the orbital configuration, a 24-thruster Manned Maneuvering Unit (MMU) [9] is strapped to its back to provide 6 DOF propulsion. The MMU model was turned off during the grasping tests. Not shown in the figures is an inertial measurement unit (IMU) composed of 3-axis accelerometers and rate gyros, which provide acceleration and rotational velocity measurements of the EVAHR body. The EVAHR will use the following computers to grasp free-floating targets in real-time onboard the KC-135: 8 Intel i860 processors connected via crossbar, 14 Transputer T800 processors networked via synchronous Transputer links, and 4 68040 processors. A VMEbus will connect the processors/ networks and sensor/actuator electronics.



Figures 3. EVAHR grasping ORU in orbital simulation.

4. EVAHR SIMULATION

Figure 3 shows testing with the orbital simulation. The arm and laser scanner will be attached to the floor of the KC-135 in flight, as in Figures 1 and 2, rather than to the body. The KC-135 simulation includes models of the a) EVAHR hardware and b) dynamics of the KC-135 environment. Models of the EVAHR hardware required to test autonomous vehicle navigation were developed initially. One of these models, the laser scanner image simulation software, continues to play a major role in vision-guided manipulation. This simulation takes advantage of the z-buffering graphics hardware associated with Silicon Graphics workstations to provide fast range image generation for polyhedral-surface objects. The model was extended to provide configurations possible in the Perceptron laser scanner - 64x256 or 256x256 images and variable field of view. The laser scanner simulation model does not try to model noise or distortions, however, other than uniform pixel noise. Preliminary tests with the actual Perceptron hardware show a maximum error of 3% in the range pixel measurements. Unfortunately, the Perceptron produces more systematic noise which is not modeled by this simulation. Target CAD models (see Figure 6) along with CAD models of the EVAHR hardware (see Figure 3), are used by the laser scanner simulation to generate simulated images.

Additional models were developed expressly to support testing of the grasping software. These include models of: a) the RR manipulator and JH4 dexterous hand, b) collision detection, and c) grasp contact dynamics. Because dynamic models of manipulators are hard to obtain and compute-intensive, we opted for an acceleration-level model of the RR 7 DOF arm and JH-4 dexterous hand. This model has since been proven in tests with the actual arm. Collision detection software was developed to detect a) desired collisions between the JH-4 palm/fingers and the target and b) collisions that the manipulator trajectory control software is attempting to avoid. The collision detection software requires that the objects be convex or be composed of convex subparts. The dynamics of each body (e.g. EVAHR, target) is computed by summing the forces and moments on each orbiting body, computing the associated translational and rotational accelerations, then integrating (at 100 Hz) to arrive at each body's velocity and position. One such translational force is gravity. When collisions between the fingers or palm of the JH4 hand and the target are detected, additional forces and moments on the target are computed and "folded into" the (orbital) dynamics state propagation. To make the computation tractable, each finger is assumed massless relative to the target and stops on contact; however, its motion is resumed if the forces/moments on the target are such that the target moves away from the finger. The forces/moments are computed one contact point at a time, even when multiple fingers are in contact simultaneously. Assuming a loss of velocity after each contact due to friction, the motion of the target is considered "stopped" when the target's total velocity falls under a threshold. This simulation has not been validated except by visual verification of the results of random test cases. Numerous cases have shown the target slip out of the grasp of the closing fingers as well as the target being trapped by them - the behavior has appeared "reasonable".

5. EVAHR SOFTWARE

The EVAHR grasp software acts as a closed-loop control system (see Figure 4), where the simulation models just described act as the plant. The vision and grasping modules are as described in 7. All are implemented in C, as are the simulation models.

Measurements of the target's pose (3D position/3D orientation) are computed by the vision modules Tracking[10] and Pose Estimation[11, 12]. The Tracker processes each image to find a rough estimate of the target's location and, if the pose estimator is ready, the contour of the object. To avoid confusing the target with the arm/hand, which may be in the image or even partially obscuring the target, the arm/hand is removed from each

image. The is accomplished by using a CAD model of the arm/hand. Pose estimates are computed by finding the best match of image features (edges/vertices) on the occluding contour to corresponding features in CAD model representations of each target (see Figure 6). The pose estimation software uses the same CAD models as the laser scanner simulation uses to draw its images and thus benefits from perfect surface geometry models of the targets. These measurements are computed at rates which vary depending upon a number of factors: the complexity/symmetry of the object, the availability of predicted poses for the image time, the degree of occlusion by the EVAHR arm/hand, and of course the computing resources available. The output rate of the pose estimator may vary from .1 sec (laser scanner minimum image interval) to several seconds.

These measurements are provided to independent Kalman filters which estimate the target's translational and rotational states. These model-based state estimators are different from those in the orbital EVAHR software, due to the different environment dynamics, and are described more fully in [13]. The rotational filter required only minor modifications, namely redefinition of the inertial coordinate system (CS). While the orbital version used an Earth-centered CS, the KC-135 version uses the aircraft's orientation at the beginning of each parabola. Thus the target's attitude is EVAHR-relative from that point on, due to the arm being fixed to the aircraft. The KC-135 translational state estimator had to be made explicitly EVAHR-relative, due to the lack of the Earth-based position correction (e.g. Global Positioning System (GPS)) that was assumed in the orbital software. The target's state is thus estimated in the EVAHR's coordinate system, using inertial measurements from the IMU and initializations/corrections from vision. For the purposes of this round of simulation testing, both the measurements and corrections were taken, noiseless, directly from the dynamics simulation. Both the translational state (position, velocity, acceleration) and rotational state(attitude, rotational velocity and acceleration) states are computed at 100 Hz for the EVAHR Manipulator Trajectory Controller. They are also archived in the World Model database as predictive feedback to the Tracking/Pose Estimation modules for images to be processed shortly thereafter.

The Manipulator Grasp Planning and Trajectory Control module[14] receives an estimated target state at 100 Hz. At 10 Hz, the Grasp Planner decides which of the multiple grasp regions available for the target is optimal for grasping. (Prior to runtime, the graspable regions on the target's surface were computed and placed in a database. These regions were found automatically based on CAD models of the JH4 hand and the target.) The Manipulator Trajectory Controller (MTC) computes RR/JH4 joint accelerations to track the motion of the chosen grasp region on the target at 100 Hz. Each cycle the MTC also computes joint accelerations to avoid a) joint singularities, b) joint limits, c) RR link-link collisions, d) RR-EVAHR body collisions, and e) collisions between the RR/JH4 and regions on the target which are not to be grasped. The MTC uses a potential field-based algorithm to compute these avoidance accelerations.

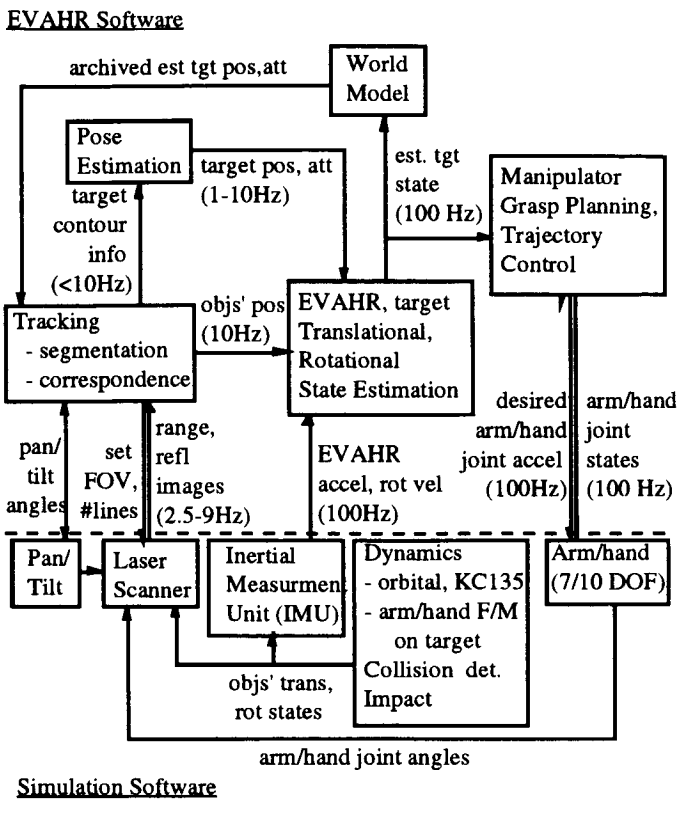


Figure 4. EVAHR/Simulation Software Architecture

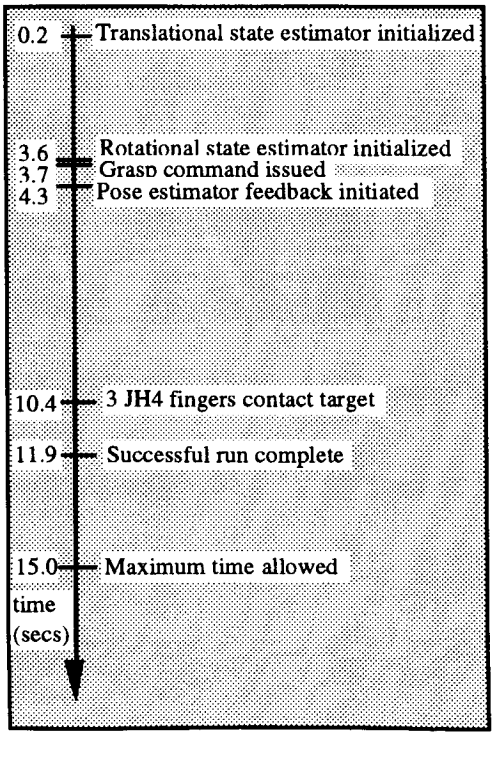


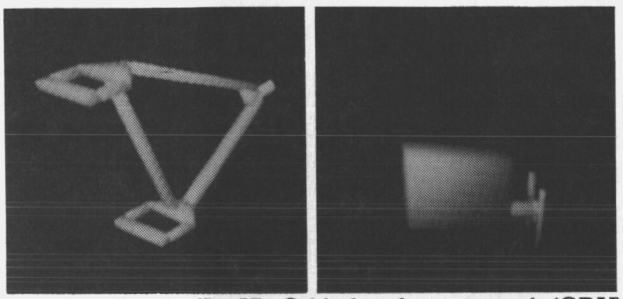
Figure 5. Avg event timeline for NBox.

The MTC takes advantage of the extra DOF in the manipulator to minimize the effects on the end-effector trajectory.

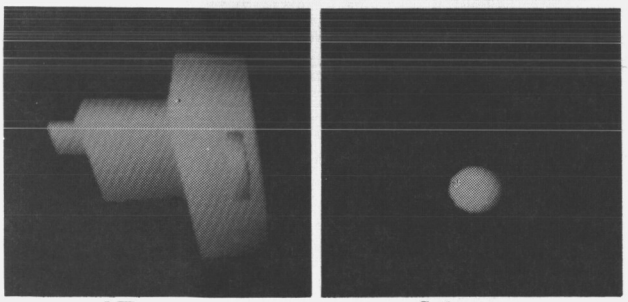
The cycle rate of 100 Hz was determined primarily by the need to control RR joint accelerations at that rate. It was also determined previously that propagation of the target's rotational state would diverge if the integration timestep were less than approximately 50 Hz.

The software was run on 2 Sparcstations and 1 Silicon Graphics 70GT for the experimental testing. Figure 5 shows a representative sample event timeline for an attempted grasp of the NBox.

Two different types of target shapes were used in testing: spherical and polyhedral. In the case of the sphere, the pose and rotational state estimation modules are unnecessary and grasping is greatly simplified as well. Therefore, polyhedral objects were used almost exclusively for testing. Three different polyhedral shapes were used (Figure 6).



Jettison handle (JettH) Orbital replacement unit (ORU)



NBox Sphere

Figure 6. Polyhedral and spherical target shapes

An additional major difference between the orbital and KC-135 software lies in the area of image segmentation. While the background of space provides an extremely convenient basis for segmenting the target in each image, the inside of the KC-135 aircraft is very difficult and compute-intensive. To minimize the computational load, a black curtain will cover the inside of the aircraft to simulate space.

7. CONCLUSION

The EVAHR software for grasping a target floating free on-orbit has been described, as well as the KC-135 simulation for testing the software as an integrated whole. The software was integrated and successful grasps were achieved, similar to those obtained in thorough, integrated testing with the orbital simulation. This result suggests that the software is ready for flight testing on-board the KC-135.

8. FUTURE WORK

A preliminary KC-135 flight test using spherical targets exclusively will be conducted in the near future. The advantage of spherical targets is that they have no rotational state and, hence, neither the pose estimation software nor the rotational Kalman filter are required. A Teleos PRISM3 stereo-vision system, instead of the Perceptron laser scanner, will be used to provide target position inputs (at > 20 Hz). The translational state estimation and grasping software modules have been reimplemented on parallel Intel i860 and Siemens Transputer architectures to achieve real-time speeds. Calibration of the arm and PRISM3 systems has been completed and pre-flight integrated systems testing is to begin shortly.

Meanwhile, modifications are being made by Perceptron to the laser scanner to improve its noise characteristics. Of primary concern is range drift, which is serious enough to prevent the sensor from being calibrated. Assuming that the problems with the laser scanner are corrected, a flight with the polyhedral targets is scheduled for sometime in the summer of 1994.

10. ACKNOWLEDGMENTS

This work was performed under contract NAS 9-17900 with the Automation & Robotics Division of Johnson Space Center. Also, the Lockheed (LESC) developers of the software modules - G. Anderson, C. H. Chien, L. Hewgill, and M. Littlefield - are as responsible for this paper as the author. Without them, these results could not have been produced. Comments and suggestions regarding the testing made by Dale Phinney (LESC) and Jon Erickson (NASA) were also greatly appreciated.

11. REFERENCES

1. C. Laugier, A.Ijetl and J. Troccaz, "Combining vision-based information and partial geometric models in automatic grasping," IEEE Int. Conference on Robotics and Automation, p. 676, 1990

2. T. Lozano-Perez, J.L. Jones, et al, "Handey: A robot system that recognizes, plans, and manipulates", IEEE Int. Conference on Robotics and Automation, p. 843, 1987

3. G. Hirzinger, et al, A sensor-based telerobotic system for the space robot experiment ROTEX, ISER '91, Second Int. Symposium on Experimental Robotics, Toulouse, France, June 25-27, 1991.

4. R. Norsworthy and L. Merkel, EVA Retriever hardware and on-orbit environment simulation requirements and design, JSC-24653, NASA Johnson Space Center, Houston, Tx, Nov, 1990.

5. R. Norsworthy, Synthetic laser range imagery using z-buffering, Summer Computer Simulation Conference (SCSC) '90, Calgary, Alberta, Canada, July 16-18, 1990.

6. K. A. Grimm, et al, An experiment in vision-based autonomous grasping within a reduced gravity environment, SPIE Proc. 1829, Cooperative Intelligent Robotics in Space III, Nov. 1992.

7. R. Norsworthy, Performance Measurement of Autonomous Grasping Software in a Simulated Orbital Environment, SPIE Telemanipulator Technology and Space Telerobotics, Vol. 2057, Sept. 93.

8. Whitman, J., "EVA crew separation dynamics final test report", NASA Technical Report JSC-23296, March, 1989

9. C. E. Whitsett, Role of the Manned Maneuvering Unit for the Space Station, SAE Technical Series 861012, July, 1986

10. M. Littlefield, Adaptive tracking of objects for a mobile robot using range images, SPIE Proc. 1829, Cooperative Intelligent Robotics in Space III, Nov. 1992.

11. C. H. Chien, A computer vision system for Extravehicular Activity Helper/Retriever, SPIE Proc. Sensor Fusion and Aerospace Applications, Orlando, Florida, April 1993.

12. C. H. Chien and J.K. Aggarwal, Model construction and shape recognition from occluding contours, IEEE Trans. on Pattern Analysis and Machine Intelligence, 11(4):372-389, April 1989.

13. L. Hewgill, Motion estimation of objects in KC135 Microgravity, AIAA Conference on Intelligent Robots in Field, Factory, Service and Space (CIRFFSS), March 1994.

14. G. Anderson, Grasping a rigid object in zero-g, SPIE Proc. 2057, Telemanipulator Technology and Space Robotics, Sept. 1993.

AIAA-94-1287-CP

OBJECT TRACKING WITH STEREO VISION

Eric Huber
The MITRE Corporation
Houston, TX
huber@mitre.org

Abstract

Task directed, active vision techniques are beginning to produce vision systems capable of providing real-time depth perception for robots. The concepts of shallow disparity filters and coarse disparity segmentation show particular promise for several reasons: they require minimal correlation search, thus they are fast; they employ redundant measurements, so they are noise tolerant; and they are not dependent on precise measurements, so they are calibration insensitive. We have extended these techniques by coarsely segmenting regions about the fixation point according to the task at hand. Relative occupancy of these regions provide cues to the approximate location of relevant features. They also provide reliable information for maintaining gaze stabilization. We have implemented these techniques on a real-time active stereo vision system to produce robust visual skills for tracking highly dynamic objects and occluding contours. These techniques are general in that they can be applied to a wide variety of objects.

Introduction

Vision has long been recognized as one of the most significant pieces of the autonomous robot control puzzle. The richness of information available in the visible spectrum makes vision unique from other sensing modalities [Mataric]. The vision research community initially endeavored, unsuccessfully, to assimilate all available data which resulted in computational overload. Much of the recent success in vision can be attributed to a transformation in thinking which has swept the vision community. Stereo vision, in particular, has seen a resurgence as the role of attentive mechanisms, foveation, and vergence in biological models has become better understood. Exciting developments in gaze control and task directed vision are now bringing long awaited goals within our grasp.

Classical Stereo

Stereo vision provides a means for depth determination through the principal of triangulation. If an object in a scene is located in the center of a stereo image pair, then that stereo system is said to be verged, or fixated, on the object.

In the more general case, features in a scene will be viewed from a variety of depths. Assuming that there is some mechanism for matching (correlating) features between the stereo image pair, the feature disparity (the relative number of pixels from the optical axis) for each image is required to determine the object location. In addition to the baseline, the angle subtended by each pixel must be known for the stereo cameras. The accuracy of these measurements is dependent on the accuracy with which calibrated system parameters are known.

Feature matching is a non-trivial problem in stereo vision. Extensive research has resulted in a number of schemes ranging from simple pixel-pixel correspondence or vertical line matching [Braunegg 90], to more sophisticated approaches which employ filtering of the images prior to matching in order to normalize the scene and accentuate its salient features [Marr-Poggio 79]. Still other schemes utilize a variety of primitives [Marapane-Trivedi 92] in order to determine correspondence. The approach taken to achieve correspondence is often driven by the characteristics of the domain in which the vision system is to be operated.

Stereo Methodology

When given the opportunity to acquire stereo and other forms of vision data, it is tempting to try to generate a complete 3-d map (reconstruction) of the environment. Early attempts at reconstruction consistently met with insurmountable difficulty due to the complexity of the real world (and its lighting conditions). Real-

Copyright c 1993 by the American Institute of Aeronautics and Astonautics, Inc. All rights reserved.

time performance was only conceivable in extremely limited domains.

When these vision systems were applied to robotic tasks such as navigation, the lack of real-time performance became most evident [Moravec 83]. Even if an accurate reconstruction were achievable in real-time, the task of "making sense" of the resultant geometry is computationally expensive.

To complicate matters, 3-D matching algorithms require precise depth information in order to converge on a correct solution. Precise data requires precise calibration which is not only time consuming to achieve, but difficult to maintain [Grimson 93] when sitting atop a dynamic robot. Finally, biological models of successful stereo vision systems do not suggest a dependence on highly accurate "calibration".

Biological models have provided some insights to the vision community which is turning away from reconstruction and towards approaches which can be managed in real-time. The prevailing attitudes held by the vision community include:

- The desire to make computation more manageable by limiting computation to what is most relevant to achieving the task at hand [Ballard 91].

- The desire to exploit regularity in the environment to allow for reduced complexity (Successfully demonstrated) [Horswill 93].

- The need to tightly couple sensor control with what is being perceived (active vision).

The last concept is common in the stereo vision community which is particularly concerned with fixation [Ballard 91] and gaze (vergence) [Coombs-Brown 91] control.

One of the most difficult problems in stereo vision is matching; however, recent techniques for reducing the computation associated with correlation have met with success [Grimson 93]. By reducing the range over which a match is likely to occur, the search space and the false match rate are reduced [Olson 93]. This approach may be extended in order that shallow disparity regions can be used for local segmentation in depth [Grimson 93]. Such approaches have only begun to be explored but show much promise.

In support of robot activities it is necessary for vision systems to provide relevant information in real-time. Even when techniques for computational simplification have been employed, a large amount of data must be processed in order to be useful to a robot. An active, real-time stereo system is still a "complex beast". Few institutions [Nishihara 90] [Marapane-Trivedi 92] have the resources necessary to assemble and program the specialized hardware required for real time stereo vision. However, it would be more difficult to simulate the interaction between a noisy, dynamic world and an active stereo vision system than to build the real thing. These are the primary reasons for the paucity of real-time stereo vision research. This will improve as computing hardware and digital cameras become faster and more accessible.

PRISM Stereo Vision

The following discussion concerns our work in active stereo vision. The objective of our R&D efforts is to provide needed visual perception skills to the mobile and articulated robotic systems we support. This stereo vision project, which has been ongoing for approximately one year, employs a PRISM vision system which will be described briefly.

The PRISM [Nishihara 84] [Nishihara 90-B] stereo vision system is the embodiment of Keith Nishihara's biologically inspired Sign Correlation Theory [Nishihara 90-A]. Nishihara's theory is an extension of Marr and Poggio's Zero-Crossing Theory [Marr-Poggio 79] . The Zero-Crossing Theory was found to be effective at extracting the salient features of an image in a normalized fashion, but the matching algorithm did not perform well in the presence of noise. Sign Correlation is the dual to Zero-Crossing contour correlation, yet it is highly tolerant to noise and preserves the intent of Zero-Crossing Theory (Nishihara 90-B).

The PRISM system employs dedicated hardware to provide spatial and/or temporal disparities in real-time. It is comprised of two frame-synchronized cameras mounted on a servo controlled pan-tilt-verge head. The camera output is digitized and fed into a custom Laplacian of Gaussian (LOG) convolver board. The LOG convolved stereo output is then buffered on a high speed, parallel sign correlation board. A 68040 processor is used to control the operation of the LOG convolver, sign

correlator, and a motion controller for the pan-tilt-verge head.

We have selected several goals for our stereo vision system which are intended to provide the greatest benefit to the robotics projects which we support. The list is representative of the goals described by the vision community in general which includes: tracking, object size and shape estimation, obstacle detection, and object recognition. In pursuit of these goals, our approach is largely based on real-time active vision principals. In addition, we are concerned with computational economy, modularity, reusability, and man-machine interaction.

Motion-Centroid Tracking

Originally, our PRISM system included some application software for performing differential motion tracking. The tracking algorithm which was provided computes a monocular disparity vector, for a single image patch, from consecutive monocular frames (temporal disparity). The disparity vector (motion vector) is then used to update pan-tilt velocity control parameters. These parameters are fed to the motion controller which keeps the tracked object in the field of view. Vergence is maintained through stereo disparity matching and is centered about the same location as the motion correlation window.

This scheme enables the system to track an object for a surprisingly long time (due in-part to the accuracy gained by the subpixel disparity measurements made possible through sign correlation) considering that disparity errors accumulate with each frame with no mechanism for re-centering on the object. Still, these errors eventually accumulate to the point where the attentive mechanism "slips off" an occluding contour of the tracked object and "gets lost" on the background. This scheme also proves to be very fragile when faced with rotating objects.

Since the differential tracking algorithm is designed to tap the strengths of PRISM, its speed is adequate, but it lacks the robustness required for our purposes. Therefore, we developed an algorithm which combats the effects of rotation and integrated disparity error by re-centering on the object. After the motion and range disparity measurements, there was only about 1/5 of a frame period left for centering oriented measurements; thus it was necessary to stay true to our goal of computational economy.

The centering algorithm (CA) we developed is simple yet robust. It takes advantage of the fact that the differential tracker typically maintains the object of interest within a shallow stereo disparity range. The advantage to limiting disparity measurements to a shallow range is that it greatly reduces the number of correlations necessary to search a given region of space. A determination of occupancy can be made quickly by checking one of these shallow regions for high (hits) or low (misses) correlation. The centering algorithm employs a grid of shallow disparity measurements (disparity grid) which can then be interrogated to find an approximate area centroid for the object. The CA simply biases the pan-tilt velocity command towards the centroid of the object acting, in effect, as an integral control term with respect to the differential tracker.

The CA described above is almost qualitative in nature. Each measurement is binary, and a number of binary results are used to drive the attentive mechanism. This approach provides several advantages: each measurement is relatively low cost allowing for several measurements per frame; a number of measurements are averaged making the system insensitive to noise; and the measurements are well distributed, providing sufficient information to track even sparsely textured objects.

As an object is tracked, its size and shape may change as its relative pose changes, revealing different portions of the object. In order for the centroid tracking algorithm to be general, it had to be designed to normalize with respect to such image transitions. Normalization is achieved through a low cost analysis of the occupancy distribution within the disparity grid. If the "hits" and "misses" are homogeneously distributed then the grid size is increased. If there are only a few "hits" and they are localized, then the grid size is decreased . The resultant behavior is that the grid constantly seeks to match the scale of the object it is tracking, a necessary precursor to maintaining center.

A beneficial side effect of the grid size normalization mechanism is that the grid tends to expand as much as possible, thus it attempts to engulf the larger, more track-able portions of an object. For example, if a motion seeking algorithm "wakes up" the tracker when it is fixated on a person's hand, the disparity grid will tend to move the fixation point up his arm and eventually to his torso. Once fixated on the torso,

the tracker is very robust and it is difficult for someone to "lose" it.

Finally the disparity grid depth of field (defined in disparity space) had to be normalized with respect to the object's distance. Otherwise, while tracking an agent which strays far away, the expanded depth of field may allow the object to blend into the background. This also requires little more than a single trig. computation.

Depending on the task at hand, the centroid tracker can be influenced to fixate on specific regions of an object. For example, when tracking a human, a small velocity control bias in the +Tilt direction causes the tracker to reach equilibrium on the head rather than the torso. We expect this skill to come in handy in the future when we will need to segment out the head and/or face for recognition purposes.

Depth Tracking

Centroid biased motion tracking has proven effective, yet expensive. In order to find the displacement of a correlation window between consecutive frames, it is necessary to tessellate (in effect) an area of the image with correlation windows. For animate objects such as humans, anticipated accelerations require a large array of correlations. Stereo disparity measurements, however, require only a one dimensional search, thus they are relatively inexpensive.

In many instances, disparity segmentation alone is adequate for locating an object. For this purpose a depth tracking algorithm was developed which again makes use of disparity regions. This algorithm utilizes a pyramid approach to search for a distinctive object by segmenting with respect to depth and cyclopean axis proximity. Again, the object centroid is used as the reference location. For tracking spatially distinct objects, the performance of the depth tracking algorithm (implemented on the PRISM system) is astounding. In fact, in this configuration, tracking accelerations are limited by the mechanics/control of the pan-tilt-verge head rather than the computational speed. Additional visual cues would be necessary to make this approach to tracking viable in cluttered environments.

Contour Tracking

For determining the shape and size of objects which cannot be contained within the camera field of view, it is useful to be able to trace, or in a more dynamic sense, track its contours. To achieve this, the author has developed a mechanism which employs methods similar to those used for tracking object centroids. The difference is that instead of avoiding object boundaries, it is desired that the disparity grid straddle them. If the system can be made to seek out and "stabilize on contours", then determining the "sense" of the tangent vector needed to track along the contour is a trivial matter.

In order to develop a contour stabilizing algorithm using stereo disparity, it is necessary to contemplate the depth transitions characteristic of these contours. The depth gradient of a contour may be positive, zero, or negative with respect to the stereo coordinate frame.

Occluding contours have special characteristics which readily lend themselves to stable tracking. A disparity grid of measurements straddling an occluding contour will have a distinct cluster (dark area) of "hits" on an object and a distinct region (light area) of "misses" off of the object edge. An algorithm which employs competing "forces" acting to keep these areas in balance has proven stable even when limited texture was available. Contour tracking is induced by moving along the perpendicular bisector of the dark and light regions.

The remaining question is "how to update the median disparity depth of field as the contour is traversed". This question is key because the disparity depth of field must be appropriately biased to ensure that it always encompasses the occluding edge. Failure to meet this criteria will not ensure that the occluding contour will be tracked accurately as its slopes towards or away from the vision reference frame. For occluding contours, the appropriate bias is found by averaging disparities near the perpendicular bisector. The trend indicated by the depth differential between these points, and points on the interior of the edge, provide a clue about the slope of the occluding contour. The disparity depth of field must be biased in the direction of this slope.

This scheme has proven to be very successful for tracing out distinct objects (spatially

separated from other objects) including cardboard boxes (poorly textured), a robotic manipulator (wires and all), and several humans. The algorithm, as implemented on the PRISM, includes some dynamic "optimization" which modulates the head control velocity based on contour detectability and smoothness. The system can now trace out an average sized human, standing seven feet away, in about twenty seconds, although, it currently cheats in order to find closure when it reaches ground level.

Interior contours present a slightly more difficult depth segmentation problem and therefore present a greater challenge for gaze stabilization. Several approaches to real-time interior contour tracking are under development.

Conclusion

The benefits of using local disparity regions for coarse grain depth segmentation has been discussed. Several successful stereo vision skills which utilize these measurement techniques have been described.

Our PRISM system is scheduled to be integrated with a Cybermotion platform. We intend to use the visual skills described above for guiding the robot to follow humans, walking at a natural pace, through cluttered environments.

Acknowledgments

Support for research reported on in this paper was provided by NASA and was performed in the Laboratories of the Automation and Robotics Division at Johnson Space Center, NASA. I am grateful for the insight and encouragement provided by Ken Baker.

References

[Ballard 91] "Animate Vision", Dana H. Ballard, Journal of Artificial Intelligence, 48:57-86, 1991.

[Braunegg 90] "MARVEL: A System for Recognizing World Locations with Stereo Vision", David J. Braunegg, MIT A.I. Lab Technical Report 1229, May, 1990.

[Coombs-Brown] "Cooperative Gaze Holding in Binocular Vision", David J. Coombs and Christopher M. Brown, IEEE Control Systems Magazine, June, 1991.

[Grimson 93] "Why Stereo Vision is Not Always About 3D Reconstruction", W. Eric L. Grimson, MIT A. I. Memo #1435, July, 1993.

[Horswill 93] "Polly: A Vision-Based Artificial Agent", Ian Horswill, Proceedings of the Eleventh National Conference on Artificial Intelligence, 824-829, July, 1993.

[Marapane-Trivedi 92] "Multi-Primative Hierarchical(MPH) Stereo System", Suresh B. Marapane and Mohan M. Trivedi, IEEE Conference on Computer Vision and Pattern Recognition , June 1992.

[Marr-Poggio 79] "A computational theory of human stereo vision", D. Marr and T. Poggio, Proceedings of the Royal Society of London, 204:301-328, 1979.

[Mataric] "Perceptual Parallelism and Action Selection As Alternatives to Selective Perception" , Maja J Mataric (MIT Artificial Intelligence Laboratory).

[Moravec 83] "The Stanford Cart and The CMU Rover", Hans Moravec, Proceedings of IEEE, 71 :872-884, July, 1983.

[Nishihara 84] "Practical real-time imaging stereo matcher", H.Keith Nishihara, Optical Engineering, 23(5), 1984.

[Nishihara 90-A] "Real-Time Implementation of a Sign-Correlation Algorithm for Image-Matching", H. Keith Nishihara, Teleos Research Technical Report 90-2, Palo Alto, CA, 1990.

[Nishihara 90-B]"PRISM-3: Real-Time Binocular Stereo and Optical Flow Measurement System" H. Keith Nishihara, (Teleos Research, Palo Alto, CA), 1990.

[Olson 93] "Stereopsis for Vergeing Systems", Thomas J. Olson, IEEE Computer Vision and Pattern Recognition Conference, 55-60, 1993.

AIAA-94-1288-CP

A SOFTWARE ARCHITECTURE FOR HARD REAL-TIME EXECUTION OF AUTOMATICALLY SYNTHESIZED PLANS OR CONTROL LAWS

Marcel Schoppers
Robotics Research Harvesting
166 Springdale Way, Redwood City, CA 94062

Abstract

We present a hard real-time software architecture which enables two kinds of safety within a single system: the safety of flexibility and robustness, and the safety of guaranteed timing. The architecture combines a) online intelligent synthesis of courses of action, with b) precise timing and very high speeds in the performance of such automatically synthesized plans. A fundamental component is a negative feedback loop from available computing resources to computational demands, which ensures both that online decisions are feasibly performable, and that the available computing resources are used to best advantage. This feedback loop allows the architecture to respond to degraded sensor systems by scheduling alternative computations without accidentally impairing the timing of other tasks; allows to respond to degraded computers by scaling back system activities to maintain minimal standards of performance, e.g. safety; and allows to dynamically exploit excess computing power for improved performance.

The architecture, though still subject to limits on the achievable robustness, can nevertheless be expected to out-perform systems in which computer power is allocated off-line. We briefly describe two robotic subsystems which can not be built safely without our architecture.[1]

1 Motivation

1.1 Two Control System Problems

Some NASA applications, such as the intelligent management of hardware faults in Space Stations and spacecraft, and the use of intelligent mechanisms (e.g. robots) about the Space Station and on Mars, require both *flexibility* and *hard real-time* execution. Two cases in point are the intelligent control of "redundant" robots and the dynamic reconfiguration of perception processing, both of which we are currently implementing.

[1]Copyright ©1993 by Marcel Schoppers. Published by the American Institute of Aeronautics and Astronautics Inc. with permission.

With only 6 degrees of freedom a robot arm can reach any posture in exactly one way, and this simplicity makes mathematical motion analysis easy. The NASA/JSC robot called the Extra-Vehicular Activity Helper/Retriever (EVAHR) has 20 degrees of freedom: 7 in each of the two arms, and 6 in the body. As a result there are an infinite number of ways for the EVAHR to get one hand to a particular posture. The extra ("redundant") degrees of freedom greatly complicate the motion control problem. Our approach to the control of such a "redundant" robot takes dynamical limits into account, validates the robot's expected motion, and also allows it to start moving immediately. This is accomplished as follows.

(1) Our Artificial Intelligence (AI) software uses a qualitative kinematic model to choose a sequence of postures for the robot to achieve [Jung and Badler, 1992]. This sequence of postures deals with robotic "redundancy" in roughly the way humans would: to reach under a bed, one gets down on one's knees, rests on all fours, lowers one's head, and stretches out one arm. Each intermediate posture is then treated as an attractive potential field. The attractive fields, together with their repulsive counterparts (obstacles) drive motion dynamics in the usual way (after [Khatib, 1986]) thus striving to avoid collisions and ensuring that the motions are compable with the robot's force and torque limitations. A super-real-time motion previewer then uses these potentials to preview the resulting motion before it is performed, and so verifies that the motion will avoid both collisions and potential wells (local minima).

(2) For this scheme to work, there must be a careful synchronization between trajectory previewing and the actual motion, lest the actual motion get into space that has not been previewed, with potentially dangerous consequences. Indeed, for given motion speeds and accelerations the previewer must be executed with a frequency lying within a limited band [Schoppers, 1993]: if the previewer is executed too frequently it won't have time to preview a fail-safe trajectory; if it is executed too infrequently the robot can accelerate ahead of it.

The preceding two paragraphs taken together signal trouble. Paragraph (1) says that we need an Artificial Intelligence (AI) program to decide, on-line, how the robot should move, by choosing a sequence of postures. Paragraph (2) says that motion-related computations must

be carefully timed. These two requirements have been incompatible for many years.

(3) Our second robotic subsystem provides a new degree of system robustness and survivability for robotic vision. We take advantage of the presence of several types of vision sensors on the EVAHR robot by dynamically selecting available sensors and perception algorithms, thus improving the probability of system survival despite sensor malfunctions.

(4) Robotic vision is part of the pipeline from sensor input through state estimation to motion control. Since data reaches the robot's effectors some time after the data was first sensed, the control system must compensate for the delay by predicting where things *will* be when the intended actions are finally performed or completed. If any part of the sensor-input-to-action-output pipeline is mis-timed, the control system's predictions will be wrong and damage may result.

Here too, the combined implications of paragraphs (3) and (4) are problematic, but for a different reason. Perception can be implemented without resorting to AI algorithms; but because the computing time required to interpret different sensors differs widely, dynamic reconfiguration effectively makes the computing time of the whole perception system unpredictable. Once again, the conflict between system flexibility and real-time execution is holding up an otherwise good idea.

1.2 The Core Problem

It is a very difficult problem to implement systems that are at once *flexible* in the sense of adaptivity and robustness, and *hard real-time* in the sense of timely execution of actions. To show why this combinations of requirements is so difficult, this subsection presents a careful analysis of each requirement.

"Flexibility" in a machine is as apparent as its makeup is subtle. The dark background against which flexibility stands out in sharp relief is the problem of domain complexity. We explain this with reference to automated diagnosis. In principle, diagnosis could be done by compiling a large decision table that associated combinations of symptoms with faults. As the number of possible faults and symptoms grows, however, the number of possible *combinations* of faults (and symptoms) grows astronomically. The programmer is inevitably overwhelmed by the complexity of the domain, and so fails to anticipate some of the possibilities and to properly understand others. The result is software that makes false assumptions and sometimes behaves inappropriately. Flexibility, then, stands out as the ability to go on behaving appropriately even when the number of possible situations grows beyond the reach of human forethought.

The fundamental justification for Artificial Intelligence (AI) algorithms is their flexibility as just defined. Flexibility may be necessary for many reasons, including: the environment may be unpredictable, or subsystems may malfunction, or the environment may be entirely predictable but very complex. In such cases, AI algorithms can do *on-line* the problem solving that could (in principle but not in practice) have been done by a programmer off-line. Not only are AI algorithms more economical in terms of programming effort, they are also safer because the responses devised by the AI software can take dynamically occurring factors into account more thoroughly, and the resulting system is likely to generate appropriate behavior for a much wider variety of the possible situations.

We now turn to the requirement of real-time execution. "Hard real-time" execution is necessary whenever hardware must be controlled safely. In general, a computation is called "real-time" when the correctness or other value of a computation depends not only on what data or action the computation produces, but also on the time at which that data or action is produced. For example, there is a particular moment in time beyond which any computing about collision avoidance is no longer useful because a collision is no longer avoidable. More often than not, the timing of computations is a critical factor in total system safety. The field of hard real-time computing research is working on ways of *guaranteeing* that the timing of computations is correct.

It would be nice if we could build systems that were both safe on account of the flexibility built into them with AI algorithms, and doubly safe because their computations were guaranteed to be completed on time with the available computing power. But this conjunction is a difficult one. The pivotal problem was first clearly described by [Paul et al, 1991] as follows:

> The timing of actions taken by a real-time system must have low variance, so that the effects of those actions on unfolding processes can be predicted with sufficient accuracy. But intelligent software reserves the option of extended searching, which has very high variance.

Because AI algorithms — like people solving difficult problems — generally reach conclusions by making plausible guesses that may turn out to be wrong any number of times ("searching"), their total execution time is highly unpredictable. To make matters even worse, AI software in control applications often dynamically varies the tasks being carried out, thus dynamically changing the structure of the computation. When there are n computations a system could execute, there are 2^n possible subsets, each with its own execution time, so that again the total system's computational load is highly variable. Both sources of unpredictability make it very difficult to be sure that the total system can perform in hard real-time.

Thus it was necessary, until recently, to choose between the safety provided by hard real-time performance, and the safety provided by on-line automated decision making. Systems such as intelligent robots or space stations, which require both, were specifically beyond the state of the art.

1.3 A General Solution

We have undertaken to provide both kinds of safety simultaneously by means of a novel software architecture. Our architecture, which has been worked out in detail but not yet implemented, exploits a specific set of hard-real-time techniques that have become available only in 1992 and 1993.

Previous attempts by others to integrate hard real-time performance and on-line automated decision making were of two kinds. One tried to force AI software into a real-time mold by limiting on-line searching, thus also eliminating most of the flexibility. The other allowed the AI software to compute as long as it liked, and tried to design the real-time part of the system to keep the whole system out of trouble until (indefinitely much later) the AI software communicated a decision to the real-time subsystem. This allowed flexible behavior until the real-time part of the system stumbled into unfamiliar territory with the AI software still thinking about the past.

Our approach is an elaboration of the CIRCA architecture [Musliner et al, 1993]. The AI software doesn't merely send new parameters to a fixed real-time subsystem, it dynamically reprograms the whole real-time subsystem, and *simultaneously* plans total system behavior to ensure that the real-time program will be able to cope with the *chosen* future. This produces such interesting phenomena as robots slowing down to ensure that their real-time programs will not be overloaded — the AI software now can shelter the real-time subsystem and can also buy itself time to think. In short, our approach imposes a negative feedback loop from excessive computational loads to less demanding system behavior. Further, the real-time subsystem is no longer cast in stone but can be made to adapt to its context, and to perform procedures that were automatically constructed.

We were careful to design our software architecture in an application independent way, so that the solution would be suitable for use in everything from robots, to space stations, to interplanetary spacecraft, to Lunar and planetary bases — with a promise of enhanced safety in all cases.

Having resolved the central flexibility/real-time conflict, we intend to demonstrate the great value of our software architecture by having it enable two novel applications, namely the dynamical control of robots with many degrees of freedom, and the dynamic reconfiguration of a multi-sensor fusion subsystem. Our subsystem for reconfigurable multi-sensor fusion will dynamically choose whatever sensors and perception algorithms it likes, and the robot's physical behavior will adjust to the dynamically changing computing load. Our subsystem for intelligent control of the motion dynamics of robots with many degrees of freedom will be a large advance on current robot control technology. It will rely on our general software architecture for proper timing of motion previewing relative to actual motion, and for hard real-time computing in the presence of AI posture planning algorithms. Both of these applications are impossible to implement safely without our software architecture. In enabling feedback from computing load to safe behavior, our software architecture will automatically and dynamically determine the maximum speed at which a robot can safely move, even when some of the robot's computers and/or sensors are malfunctioning or disabled. As a result our architecture will also support sensor processing reconfiguration in a completely safe and general way. Both modules will demonstrate the value of our architecture for improved adaptiveness and survivability of complex control systems.

1.4 Our Approach In Perspective

The Artificial Intelligence (AI) community has come to address the requirements of real-time systems in three ways [Durfee, 1990]. When building a system that must act in real-time as well as reasoning, one can:

- Subject the AI component of the system to hard deadlines (*e.g.* anytime algorithms). Under time pressure, this results in truncation of intelligent function.
- Allow the AI component to think freely, make the real-time subsystem responsible for total system safety, and have the AI component re-parameterize the real-time subsystem with whatever guidance the AI subsystem can produce in time. Under time pressure, this results in intelligent function being left far behind the rush of events.
- Refuse to subject the AI component of the system to hard deadlines, but let the AI component dynamically reprogram the real-time subsystem with a program realizing *a discrete-event control law that preserves closed-loop stability with sufficient robustness for the period of time in which the AI subsystem is deciding what to do next.* This approach remains functional even under time pressure — almost by definition.

We regard the NASREM architecture [Albus et al, 1987] as embodying the first approach, and the architectures of Bonasso [Bonasso, 1991; Bonasso and Slack, 1992] and Gat [Gat, 1992] as embodying the second. We favor the third, in which we have imposed control-theoretic criteria upon the ideas of [Musliner et al, 1993]. The resulting architecture has the following advantages:

1. the AI software can remain intelligent and flexible;
2. the real-time subsystem need not be programmed (at system design time!) to be competent against all possible contingencies;
3. the AI software can reprogram the real-time subsystem to prevent computing overloads before they happen (e.g. with slower motion), thus pro-actively buying itself time to think;
4. the program down-loaded into the real-time subsystem can be as flexible as the AI software can make it (e.g. in a robotic application the maximum safe speed can now be a function of computational load, and so will be higher than the worst-case limit nearly all the time, allowing substantial performance gains);
5. the real-time subsystem can be a small operating system, time-slicing tasks or threads with widely differing frequencies, in contrast to the many mobile robot controllers in which all computations are executed with the same frequency.

2 Architecture for Flexible Real-time Control

2.1 Description of Architecture

Figure 1 shows the architecture we have designed on the preceding foundations; this Figure will serve as a guide for the ensuing discussion.

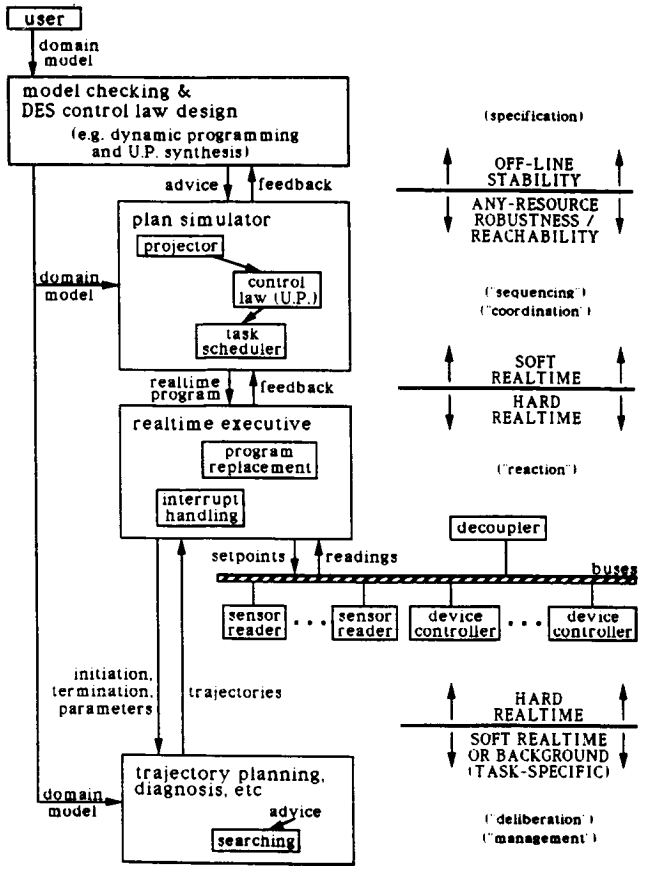


Figure 1: Architecture Diagram.

There is assumed to be a fixed control program which, in our architecture, is split into two parts. One part, usually called the control law or plan, is modified so that its execution or interpretation affects only a simulation model of the controlled system. The other part determines where simulation should begin and end, and is called the "plan simulator".

The basic idea is that the plan simulator chooses a set of initial and goal states, simulates the plan or control law over several possible futures, collects the real actions appropriate to the anticipated states of the controlled system, and so dynamically constructs a small real-time program, which it downloads to the real-time executive. From then on, the real-time executive takes care of communicating with the decoupler, sensor readers, and device controllers, in hard real-time. This mitigates time pressure on the plan simulator, which can proceed with the construction of a new real-time program. This separation of timing concerns is indicated in the architecture diagram by the dividing line between soft real-time and hard real-time.

Each new real-time program specifies: a set of functions implementing actions on the controlled system; a set of functions which test the estimated state of the controlled system; the conditions under which each action is appropriate; and maximum allowed delays between satisfied conditions and appropriate reactions.

In practice, each real-time program is a set of tasks or threads[2] that can be run, suspended, resumed, and so forth, under control of the real-time executive. Threads may be periodic, being executed cyclically and with a definable frequency; or they may require execution only sporadically (e.g. interrupts), in which case there may be a maximum allowable delay between the time the thread becomes relevant and the time it is actually executed.

The real-time executive comes with a "schedulability check" designed specifically for that real-time executive. It is used by the plan simulator to test whether each new real-time program is indeed executable in hard real-time. A successful schedulability check absolutely guarantees that no thread will miss any deadline specified for it. A failed schedulability check tells the plan simulator to design a less demanding real-time program, and if necessary, to choose a less demanding future.

The "actions" performed by the real-time executive do not themselves drive effectors or sensors. Instead, they send on/off signals and parameters to the effectors' and sensors' servo/driver loops, which often run on dedicated microprocessor chips.

Just as servo loops may take extended amounts of time to reach a setpoint, task-specific AI modules may take extended amounts of time to make a decision. Just as the real-time executive is controlling sensori-motor servo loops by updating their parameters and turning them on and off, it also turns task-specific AI modules on and off according to the needs of the moment. That's why Figure 1 shows a typical task-specific AI module *below* the real-time executive, with the device controllers.

The real-time executive, plan simulator, servo loops, and all AI computations are threads which operate in parallel by default. The fraction of computing power (CPU cycles per second) available to each thread is carefully decided by the plan simulator and enforced by the real-time executive. That is, the real-time program constructed by the plan simulator may include actions that adjust (even to 0) the frequency with which certain threads are executed.

As a special case, the real-time program may set even the plan simulator's execution frequency to 0. Even though the real-time executive depends on the now-stopped plan simulator for updated real-time programs, this situation does not mean that the system is permanently stuck with the existing real-time program, since the existing real-time program is conditional, and will have been designed so that new circumstances cause the execution of actions that allow the plan simulator to resume. (If the real-time executive's control over AI modules is reason for Figure 1 to show the latter below the

[2] A thread is a function that has its own piece of program stack, so that execution of the function can be suspended and resumed independently of the execution of other similar threads.

former, perhaps the plan simulator should also be shown below the real-time executive.)

Executed as "background" threads, AI modules can take as long as they like, and provide the system with the full power of on-line intelligent reasoning without endangering the operation of time-critical system elements. At the same time, the plan simulator can ensure, by proper design of the real-time program, that such AI modules are not left behind the rush of events: if the decision of an AI module is crucial to future activities, the plan simulator can program the real-time executive to take actions that delay the on-coming future. This resembles the idea of designing a "reactive subsystem" to ensure total system safety while the AI software thinks (see above), with the important differences that in our case (a) the "reactive subsystem" is not fixed at system design time, and (b) the AI software's thinking may be aborted and redirected by new events.

2.2 Control Theoretic Requirements

In our architecture,

> The automatically constructed real-time program is a partial control law. The real-time executive, in its performance of that control law, functions as a controller.

The hardware devices being controlled will in general have dynamical behavior. Ideally we would like the real-time program to take care of all hard real-time processes, so that the plan simulator itself does *not* have to worry about real-time deadlines. To this end we qualify the CIRCA approach with a stipulation that the real-time program must drive the controlled system into a set of goal states, and then must maintain the controlled system within that set of goal states for an indefinite period of time. Such behavior on the part of a controller is known as "closed-loop stability." For the case of discrete-event control systems, we define it as follows:

> A discrete-event system is *closed-loop stable*, with respect to a set of goal states, if the controller is able to drive the controlled system so that it actually reaches a goal state in finite time, and then (in the absence of disturbances) stays within the set of goal states indefinitely.

(Of course, along the way toward a goal state the real-time program may also decide to do nothing for a while, knowing that a device will do a desirable thing without being forced to do so.) In sum, we require, as a key property of our architecture, that

> Each automatically synthesized real-time program (discrete-event control law) must make the controlled system closed-loop stable.

When the controlled devices are subject to disturbances (i.e. dynamics that are unpredictable) the disturbances may throw the system out of the set of goal states, and we expect the real-time program to drive the controlled system back to a goal state. This too is a familiar notion in control theory:

> A control law is *robust* to the extent that it can keep the controlled system closed-loop stable despite the occurrence of disturbances and unmodelled dynamics.

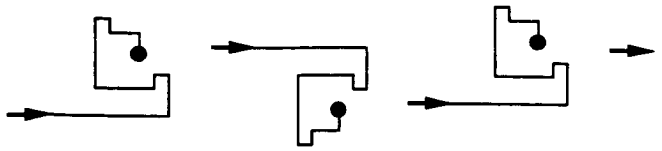


Figure 2: Controlling via Stable Subdomains.

Thus we desire that our automatically synthesized real-time program should be as robust as possible.

Figure 2 depicts the plan simulator's repeated revision of the real-time program, along with the latter's effect upon the controlled system. The closed-loop stability of the controlled system is represented by the "attractor states" into which the system eventually settles. The sequence of attractor states represents the plan simulator's repeated selection of goals. The robustness of each real-time program is represented by the fact that closed-loop stability is achieved despite unpredictable meanderings. The meanderings are rectangular rather than smooth, to emphasize that the real-time program encodes a discrete-event control law, not a continuous-variable control law.

In constructing a real-time program, the plan simulator must choose goals and must anticipate enough possible futures to achieve closed-loop stability and robustness. In practice, in discrete-event control applications, finding a set of goal states that allow closed-loop stability is much easier than in continuous-variable control applications. Cups can be made to stay on tables, doors can be made to stay open, buildings can be made to last, and so on. In AI the standard counter-examples to such stabilities are "spontaneous processes" and the actions of other agents, but remember that for our proposed architecture to work, the real-time program need only be *closed-loop* stable. This means that the devices being controlled can act to prevent unwanted interference. Thus the real-time program could prevent other agents from closing doors it wanted kept open, and so forth. Under these conditions, discrete-event closed-loop stability is not hard to find.

Achieving robustness is more difficult, especially for discrete-event control applications. The problem is that unpredictable or unmodelled disturbances may throw the controlled system into so many different states that it may be impossible to synthesize a real-time program that can respond to all of them within the CPU and/or memory resources provided by the real-time executive. Consequently, making a discrete-event control law robust is a fine art. The best that can be done is to include "important" disturbances within the domain model, and further, to include into the model such information as the likelihood, outcome, and severity of a possible disturbance. The availability of that information will allow the plan simulator to decide on-line which disturbances should be anticipated by the real-time control program being constructed, in order to make that program as robust as possible within the available resource limits.

The properties of closed-loop stability and robustness, which must hold of the real-time program being down-

loaded to the real-time executive, are shown in the architecture diagram (page 4) as being the responsibility of the plan simulator. There we have referred to "any-resource" robustness to indicate that, once closed-loop stability has been built into the real-time program, the plan simulator may apply whatever resources are left over towards improvements in robustness.

There remains the question of what should happen if, despite the construction of a feasibly robust real-time program, the controlled system makes a low probability transition into a state for which the real-time program has no response. A number of solutions are possible. In the CIRCA approach that problem was passed back to the plan simulator. Since that solution makes system stability dependent on the response speed of the plan simulator, we distrust it. An alternative is to have the plan simulator design the real-time program so as to anticipate all device states that must be controlled within some time horizon, or within hard deadlines; in this way the plan simulator will have a minimum amount of time with which to build a replacement real-time program, and will need to deal only with soft deadlines. This alternative works only if the plan simulator's model of the domain dynamics is correct. Our proposed solution is to have the real-time program include alternative subprograms, along with instructions for their contingent deployment. This solution can work if the real-time executive's execution-time resource is more constraining than its memory space resource — and that is likely.

2.3 Relation to 3-Level Architectures

The architectures of [Firby, 1989], [Bonasso, 1991; Bonasso and Slack, 1992] and [Gat, 1992] specify three "levels", namely

1. a "reactor" containing servo loops, safety reactions, and behaviors;
2. a "sequencer" for deciding which specific activities are needed both now and in the near future;
3. a "deliberator" which contains AI software for planning, diagnosis, metalevel reasoning, and so forth.

These three levels resemble our hard real-time executive, our soft real-time plan simulator, and our non-real-time AI modules, respectively. None of the cited authors took timing seriously, however. In the reactors of their mobile robots, reaction threads for avoiding collisions run every so often, but there is no guarantee whatever that collision avoidance will always be performed in timely fashion. Their programs may work properly for very long times, but ultimately it is impossible even to know whether the robots have ever endured worst-case logjams of threads competing for execution time. Hence we have replaced their reactors with our hard real-time executive, which takes timing very seriously indeed.

A second important difference is we have changed the locus of control. In both [Firby, 1989] and [Bonasso and Slack, 1992] control was hierarchical: the deliberator was on top and in control, deciding what the sequencer should do, and the sequencer decided what the reactor should do. This was changed by [Gat, 1992], who put the sequencer in control and reduced the deliberator to computing task-specific parameters like motion trajectories. The major reasons for this were that (1) reaction should not be kept waiting for deliberation, and (2) old deliberations may be irrelevant to new situations, so the sequencer should get to initiate and terminate deliberations. The locus of control changes again for our architecture: we are moving it all the way down into the reactor (a.k.a. the real-time executive). The reasoning for this is that the results to be computed depend on the situation at the time; the amount of computing power to devote to hard real-time, soft real-time, and background threads depends on what needs to be computed, and hence on the situation at the time; but the situation can change very rapidly, hence the allocations of computing power and the threads being executed may also have to change very rapidly; and so the changing allocations and the initiation and termination of all threads (including deliberations) must be handled by the real-time executive. ([Kaelbling and Rosenschein, 1990] also have a fast reactive system determining which deliberations are relevant and for how long.)

Third, our real-time executive is a real-time microkernel running threads at multiple frequencies, not a single loop that performs a fixed list of functions all at the same frequency.

3 Implementation of the Real-time Executive

Any real-time subsystem adopted as the basis of our architecture should provide the following minimum capabilities:

R1: insulating the processing time allocated to hard real-time threads from the cycle-stealing desires of all other threads.

R2: testing, as part of program execution and without modifying the set of currently executing hard real-time threads, whether a proposed new set of hard real-time threads is such that all its deadlines can be met with the available processing time.

R3: conditionally switching thread subsets on and off, so that the real-time executive is effectively executing a conditional real-time program.

R4: allowing dynamic modification of the hard real-time program (the set of hard real-time threads and the switching logic) while still meeting all deadlines before, after, and during the modification.

R5: ability to do **R1-R4** when the set of hard real-time threads contains both cyclic (periodic) and sporadic (aperiodic) threads.

These requirements can be met in a variety of ways. The alternatives are

1. multiprocessor schedulers, which are the most powerful but expensive in proportion;
2. earliest-deadline-first schedulers, which are the most user-friendly;
3. rate monotonic schedulers, which are the easiest to build.

We believe that our architecture could be implemented equally well on top of any of these options. However, the budgetary constraints on our work incline us away from multiprocessor schedulers (while yet we also believe that for such complex systems as space stations or lunar bases, multiprocessor schedulers are by far the best choice). The best fit with our work on the EVAHR robot is an earliest-deadline-first scheduler, since such schedulers facilitate dynamically changing thread frequencies (for synchronizing computing with robotic motion) and conditional schedules. Our intended implementation is similar to that of [Ramamritham and Stankovic, 1984] with the schedulability check of [Jeffay, 1992].

4 Discussion

4.1 Benefits of the Architecture

The final justification for our architecture consists of three words: safety, generality, and economy. Safety: because the timing of program execution is taken so seriously that even intermittent timing problems can not survive, and because the architecture establishes a feedback loop from computing load to graceful degradation of task performance. Generality: because the system can contain on-line intelligent reasoning, can enact the results of such reasoning in hard real-time, can dynamically replace the system's hard real-time reactions, and can judiciously control its own use of computing power. Economy: because a fixed amount of computing hardware can be time-shared and carefully allotted. Here we elaborate on just a few of these benefits; the complete list appears in [Schoppers, 1993].

- Spatial synchronization. The ability to dynamically modify the frequency of thread execution allows to synchronize computing with motion through space. With collision checks being made at regular distance intervals, slower motions require less calculation. Knowing such facts about itself, an intelligent real-time system can reduce the speed of its motions *for the sake of* reducing the processing power devoted to motion-related threads.
- Graceful degradation. The ability to scale back its operations as necessary to ensure timeliness eliminates the need to design to an imaginary worst case scenario, because there is no longer a sharp performance cliff that the system can fall off in unpredictably disastrous ways.
- Reconfigurability. A survivable system must have several ways of achieving the same result. When the sensor normally used to deliver a given datum malfunctions, another can be used. Since the computing time required to interpret different sensors differs, such result-compatible reconfiguration is only safe in systems (like ours) that can plan their behavior to match their planned computing load.

4.2 Limits of Robustness

Despite our concern for hard real-time and for dynamically achieved robustness, some kinds of mishaps can still happen (of course). If enough sensors malfunction, a robot will be unable to see new dangers approaching, so cannot be held responsible for avoiding them. Similarly a robot may, for the sake of getting its job done, have to place itself in situations that would be dangerous if the robot's computers suddenly died. In all other cases, however, including robotic inability to go on sensing objects it already knew about, as well as computer failures, our software will be aware of the potential mishaps and will continuously and intelligently redesign the robot's behavior specifically to optimize first the safety, then the performance, of the robot in its surroundings.

4.3 When Is Hard Real-time Important?

The development of our architecture was driven primarily by concern for hard real-time response despite the presence of AI software. It is unclear to many people why timing should be taken so seriously. The most common objections are (1) Couldn't we hand-code a fixed layer of real-time reactive behaviors that take care of everything (e.g. collision avoidance) while the AI software is thinking, and (2) Couldn't we make sure that the real-time software works, by means of a test-debug cycle? Sometimes yes, but also sometimes no.

Objection (1) is usually raised by people who have programmed wheeled mobile robots on earth. For such robots there is a small repertoire of actions that can ensure robotic safety, e.g. slamming on the brakes or moving away from impending collisions. However, as soon as either the robot or its environment becomes more complex, a fixed "reactive safety layer" no longer suffices. One example is the Adaptive Suspension Vehicle (ASV) [Payton and Bihari, 1991], which was the size of a bus, with six legs that were each 6 feet high at the hip. Maintaining stable balance while keeping the legs away from each other and while switching between gaits required a super-real-time motion planner. For the ASV, even stopping was so complicated that no predetermined set of "reactions" could have sufficed. Alternatively, more complex environments also prevent a hand-written safety layer, since such a layer must assume that its actions are easily reversible and will not themselves lead to new dangers. On orbit, however, a free-flying robot's action to avoid a collision will move the robot into a new orbit from which it may be both time-consuming and fuel-consuming to return, and on which it is still flying at approximately 20,000 miles per hour. In general it is not true that all robots can be kept safe forever with a fixed set of hand-coded reactions.

Objection (2) cannot be sustained if a thread's missed deadline can lead to loss of human life. Since experienced programmers know better than to claim that they've found "the last bug", and since concurrent software is worse than most, the test-debug approach may well yield life-threatening software [Stankovic, 1988]. The risks can be diminished by applying existing hard real-time scheduling research. Objection (2) is also rebutted if a system's worst-case computational load is several times higher than the average load. For example, setting a robot's top allowable speed to avoid timing problems under a rare scenario will also limit the robot's performance at all other times. Our architecture allows to

adapt the robot's top speed to current computational loading. Here too it helps to take hard real-time seriously.

Acknowledgements

We thank Peter Bonasso for stimulating debates. The architecture described herein was elaborated with the support of NASA JSC under SBIR Phase 1 contract NAS.9-18861.

References

[Albus et al, 1987] J. Albus, H. McCain and R. Lumia. *NASA/NBS standard reference model for telerobot control system architecture.* Technical Report 1235, National Bureau of Standards, 1987.

[Bonasso, 1991] R.P. Bonasso. Coordinating perception and action with an underwater robot in a shallow water environment. *Proceedings of the SPIE Conference on Sensor Fusion* IV, SPIE volume 1611, pages 320–330, 1991.

[Bonasso and Slack, 1992] R.P. Bonasso and M. Slack. Ideas on a system design for end-user robots. *Proceedings of the SPIE Conference on Cooperative Intelligent Robotics in Space* III, SPIE volume 1829, pages 352–358, 1992.

[Durfee, 1990] E. Durfee. A cooperative approach to planning for real-time control. In *Proc DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 277–283. Morgan Kauffman, 1990.

[Firby, 1989] R.J. Firby. *Adaptive Execution in Complex Dynamic Worlds.* PhD thesis, report RR-672, Dept of Computer Science, Yale University (1989).

[Gat, 1992] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. *Proc AAAI National Conf*, pages 810–815, 1992.

[Jeffay, 1992] K. Jeffay. Scheduling sporadic tasks with shared resources in hard real-time systems. *Proc IEEE Real-Time Systems Symposium*, pages 89–99, 1992.

[Jung and Badler, 1992] M. Jung and N. Badler. Human-like agents with posture planning ability. In *Cooperative Intelligent Robotics In Space III*, SPIE's OE/Technology '92. Boston MA, November 1992.

[Kaelbling and Rosenschein, 1990] L. Kaelbling and S. Rosenschein. Action and planning in embedded agents. In P. Maes (ed), *Designing Autonomous Agents*, pages 35–47, 1990. MIT/Elsevier.

[Khatib, 1986] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Internat'l Journal of Robotics Research* 5:1 pages 90–98, (1986).

[Musliner et al, 1993] D. Musliner, E. Durfee and K. Shin. CIRCA: a cooperative intelligent real-time control architecture. *IEEE Trans SMC* 23:6, 1993 (to appear).

[Paul et al, 1991] C. Paul, A. Acharya, B. Black and J. Strosnider. Reducing problem solving variance to improve predictability. *Communications of the ACM* 34:8, 1991.

[Payton and Bihari, 1991] D. Payton and T. Bihari. Intelligent real-time control of robotic vehicles. *Communications of the ACM* 34:8, pages 48–63, 1991.

[Ramamritham and Stankovic, 1984] K. Ramamritham and J. Stankovic. Dynamic task scheduling in distributed hard real-time systems. *IEEE Software* 1:3, 1984.

[Schoppers, 1993] M. Schoppers et al. *Robotic Whole-Body Dexterity and a Software Architecture for Robotic Task Performance in Uncontrolled Environments.* Phase 1 SBIR Final Report under NASA solicitation 92-1, contract NAS 9-18861, from NASA Johnson Space Center, Houston TX, 1993.

[Stankovic, 1988] J. Stankovic. Misconceptions about real-time computing. *IEEE Computer* 21:10, 1988.

AIAA-94-1289-CP

ALL FEASIBLE PLANS USING TEMPORAL REASONING

Debasis Mitra* Rasiah Loganantharaj†

{dm, logan}@cacs.usl.edu
Automated Reasoning Lab
Center for Advanced Computer Studies
University of Southwestern Louisiana
P.O. Box 44330
Lafayette, L.A. 70504

Abstract

In this paper we have discussed some of the issues involved in planning utilizing a temporal reasoning system. One of the advantages is that of being able to handle incomplete information. In these circumstances, there may be multiple plans available for achieving a task. Using an algorithm, designed recently by us, generating all feasible plans may be practicable in most of the instances, although the problem is NP-complete. The significance of having all feasible plans in a plan data base is quite important. We have also discussed these issues here.

I. Introduction

Temporal reasoning is becoming an important tool for planning[4, 5, 11]. Although it is not very easy to represent planning problem as a temporal reasoning problem, the advantage of doing so is immense. In this paper we have discussed one of such advantages. Temporal reasoning allows one to represent incomplete information between operations and other primitives in a planning problem. This is a step forward from blocks world problem towards realistic planning. Under such uncertainty, there may be more than one plan which is feasible. Using an algorithm devised by us[8], all such feasible plans can be found out. The significance of the availability of such a complete plan data base is discussed in this article.

Interval based temporal reasoning scheme gives planning activity the advantage of having some parallely executable operations. In this paper we have used this interval-based temporal representation scheme. The price of having higher expressiveness in interval-based scheme is that the problem of handling incomplete information is NP-complete. Our algorithm is an efficient one, and under most of the circumstances it should be able to handle the problem in acceptable time, although the worst case growth rate remains exponential.

II. Planning and Temporal Reasoning

Planning is the task of choosing a subset from a given finite set of operations and ordering them in time. Each operation has a set of precondition and postcondition states of the world. Preconditions are states required for an operation to become executable, and postconditions are states created by its execution. Given a set of *start* conditions and a set of *goal* conditions, the problem of planning is to see that chosen sequence of operations change the states of the world from start conditions to goal conditions. This perspective of planning is akin to a state change-based point of view in temporal reasoning, as in situation calculus. A dif-

*Graduate Student
†Associate Professor

Copyright © 1993 American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

ferent approach had been taken by Allen et al[2] about planning where they have tried to formalize planning with more explicit temporal reasoning. According to the conventional view, temporal identities, both operations and states of the world (or *fluents*), are related to each other in such a way that the end of one is the starting point of the other. In interval based temporal reasoning scheme this latter temporal relation is the 'meet' primitive[1]. There are 13 such other primitive relations feasible between two time intervals.

Classical planning, based on situation calculus, suffers from two shortcomings in their application in the real world. Firstly, it can not tackle a problem when two operations need to be performed at the same time to achieve an objective[3]. For example, to open a door one needs to turn door knob and push the door at the same time. This can not be easily represented in conventional planning schemes. Even under the circumstances where such parallel operations are not essential, such plans may execute faster (subject to the availability of sufficient resources for parallel execution of operations). Secondly, any incompleteness of information about the real world can not be represented in classical planning scheme. For example, one can make a phone call 'before' the class or 'after' the class, this flexibility of information can not be represented there. An interval based temporal reasoning scheme is capable of handling such situations.

In the interval-based temporal reasoning scheme, each of the fluents and operations are considered as an interval in time. They lie on a unique non-branching time line, and so, each of them is temporally related to all the other ones. A single primitive relation as a temporal relation between a pair of temporal entities indicates definite information between those two temporal assertions, whereas a disjunctive set of primitive temporal relations indicates incomplete information. In the next section, the scheme for interval based temporal reasoning will be discussed.

II. Interval-based Temporal Reasoning

Suppose the following set of information is given in the context of petroleum exploration.
GS = gravity survey
SS = seismic survey
DA = drilling activity
PA = production activity
ER = enhanced recovery

Apart from these operations there are following fluents which are affected by the operations. [1] Surveys are done when there has not been any survey(NS) done on the area. Seismic survey produces subsurface geological knowledge(GK), it also produces huge noise(NO) caused by the artificial seismic explosions. Gravity survey can not be done during such noise. Drilling activity requires geological knowledge about the area, although it may make geological knowledge invalid. It also makes drilling wells available(DW) which are required by production activity. Production activity produces data about the reservoir(RD) which is necessary for enhanced recovery. Enhanced recovery is done for maximum production from a reservoir(MP). Actual temporal relations between them are given below.

1. NS –{finished-by}–> GS [2]
2. NS –{finished-by}–> SS
3. SS –{equal}–> NO
4. NO –{before, meet, after, met-by}–> GS
5. SS –{overlap}–> GK
6. GK –{during-inverse, finished-by, overlaps}–> DA
7. DA –{starts}–> DW
8. DW –{during-inverse, finished-by, overlaps}–> PA
9. PA –{overlaps, statrs, equal}–> RD
10. RD –{starts, equal, overlap, meet, before}–> ER
11. ER –{meets}–> MP

[1] The information given here about petroleum production is not necessarily realistic. Preconditions and postconditions are simplified to a great extent.

[2] For exact semantics of the 13 primitive temporal relations see[1].

Given these constraints, one may need to plan the activities for achieving maximum petroleum production from a reservoir, a state of MP, starting from a state of NS(no-srvey). For example, a trivial plan could be a serial ordering of the operations GS, SS, DA, PA and then ER.

Each of those operations or fluents is an interval in time. Some of the relations may not be consistent with respect to the others. A temporal reasoner's primary job is to find out whether any consistent scenario is feasible or not, by propagating above constraints all over the temporal data base. In the context of planning, this means whether there exists a plan or not. If there exists a temporally consistent scenario, or a *plan*, then finding one such instance would be the next task of a reasoner. Formalizing planning problem as an interval constraint propagation problem, is being addressed to in [2]. According to this formalism each interval is represented as node in a constraint graph, and disjunctive temporal relations as labels on the arcs between these nodes. The temporal constraint graph (TCN) is a complete graph because every temporal interval (we call it as *t-node*) is related to the other t-nodes, even if there is no specific information about how they are related. This complete lack of information is represented as disjunction of all 13 primitive relations (termed as *tautology*).

Any temporal constraint propagation algorithm systematically eliminates primitive relations from the labels on arcs, which are inconsistent. A global consistency algorithm[3] eliminates all such primitive relations which can not form a consistent scenario for the temporal assertions in the network. In such case, any primitive relation on any label can take part in at least one consistent scenario. Such a labelling is called *minimal labelling*. If during propagation any label gets all its primitive relations stripped off, having a *null* relation, then it implies that the two end nodes can not have any consistent labelling between them. This in turn implies the given constraints were inconsistent with respect to each other, and any consistent temporal scenario can not be formed.

Interval-based temporal reasoning increases expressiveness of planning. But the main problem with this scheme is that the problem of checking global consistency is NP-complete. There are approximate algorithms to address the problem and have inexact solutions[1, 10]. But generating a plan needs a temporally consistent model. Lack of this aspect was one of the weaknesses of the original work in this line by Allen et al[2]. In their scheme the arcs of the network will be left with *consistent* labelling, which is still a disjunctive set. Making a total order of the nodes is not feasible from such labelling, which is demanded by a planning problem. To manage the efficiency issue they have proposed a clustering approach. This would keep the network size under control so that run time of the algorithm is less affected by the increase in problem size. In this scheme all intervals should be clustered into a few groups based on some reference intervals, and propagation algorithm runs separately within each group. Although in some problem domains such hierarchy of reference intervals may be inherent, in many domains, like planning, this approximation may be impractical.

III. Finding All Feasible Temporal Scenarios

A global consistency algorithm finds out a globally consistent models or all consistent models as a side effect while trying to determine global consistency of a network. Any trivial backtrack algorithm can do this work in exponential time. It is important to devise heuristics to make it efficient. First heuristic based global consistency algorithm was proposed by Valdes-Perez[12]. There, the algorithm was not written well enough to be efficiently implemented. Recently Ladkin et al[6] and we [8] have come up with two different heuristic based algorithms for solving global consistency problem.

Ladkin et al's algorithm randomly picks up a singleton relation from an arc and runs an

[3] Global consistency algorithm checks for consistency of constraints all over the network, in contrast to a local consistent algorithm, which checks for consistency of every subnetworks of a fixed size.

approximate algorithm to update relations on the other arcs while checking for consistency of the picked up primitive relation. If picked up relation is found to be inconsistent the algorithm backtracks over the set of disjunctive relations on the current and previously picked up arcs, otherwise it goes ahead with the next arc. The algorithm terminates if a singleton labelling is found for all arcs, generating a consistent model. It may also terminate by backtracking up to the arc, which was picked up first, implying no consistent scenario is available. Experimental results, presented by them, provides us with a confidence that the global consistency is a not a very difficult problem in average case, although its worst case behavior is NP-complete[4].

Table 1: Forward Pruning Algorithm

```
For node number i=1 to N do
   pickup an old singleton model of size (i − 1);
   for node number j=1 to i − 1 do
      pickup singleton relation from label on
      arc < i, j > and update labels on all arcs
      from < i, j + 1 > to < i, i − 1 > with respect to
      this singleton and arcs in old network;
      if updation fails on some arc
         if there is any more singleton left on this arc
            pickup next singleton and proceed
         otherwise backtrack to previous arc;
      if backtrack exhausts up to the first arc
         if a model was found
            save it in data base of partial models
            (of size i);
         otherwise return failure;
   if arc < i, i − 1 > is reached force backtrack;
```

The algorithm proposed by us[8] is based on a pruning heuristic where inconsistent relations are eliminated systematically. We call it *forward pruning* algorithm, because it prunes labels on all *forward* arcs in a preassigned order. A lose version of the algorithm is given in Table 1. There is reason to believe that such systematic working should improve the efficiency. Another feature in this algorithm is that the nodes are also systematically picked up one by one. At each stage of such addition of a node in the graph, a set of feasible models are generated. Then the next node is attempted for addition to each of these models. This model based approach makes it possible to generate all feasible consistent scenarios at a much lesser cost. In the worst case, the number of models itself may be exponentially growing with the number of nodes. Some preliminary experimental results with this algorithm have shown that in reality this growth rate is not very high, and manageable.

Efficiency of the forward pruning algorithm will also be aided with a fast preprocessing algorithm, which we have devised recently for incremental addition of nodes to temporal constraint graph[9]. The incremental and model-based approach will also make a *plan data base* available, in which new *operations* can be added one by one. Then new plans can be generated with respect to these newly added operations, or any newly added information. Significance of having all feasible plans will be discussed in the next section.

IV. Significance of Having All Feasible Plans

Availability of all feasible plans allows one to have a choice of picking up the best plan according to some criteria. For example, in the previously mentioned case of petroleum exploration, one can find out a plan whose execution will take lesser time than most other plans. This can be done by choosing a plan where maximum number of 'overlap' relations occur (for parallely executing those operations)[5]. This procedure can also be automated by assigning some order of priority on the primitive relations on each of the labels on arcs. The resulting models, or plans, will then also be ordered by the system accordingly.

Normally a planning activity involves reaching a preassigned goal state. Most of the current planning systems are based on this objective. A temporal reasoning-based planning sys-

[4] A very good example of such cases where a NP-complete problem is addressed comfortably in most instances, is the case of linear programming using simplex algorithm.

[5] Thus, a *good* plan there, is SS after GS, SS overlapping DA, DA overlapping PA and PA overlapping ER.

tem, as described in this article, need not be goal driven only. Using a global consistency algorithm one can generate all possible scenarios, which allows one to do temporal projection. Then one can choose appropriate goal state and a corresponding plan, from those future scenarios. Thus, planning becomes a subproblem of temporal projection problem[7]. As far as we know the implication of this is yet to be studied.

In a dynamic situation where an agent is executing a plan unknown by another agent, if all feasible plans for the first agent are available, then the second agent can recognize the first one's currently executing plan by observing its operations executed so far. The plan recognition here becomes a problem of matching actions executed in past, or a partial plan, with the plans in the plan data base, and finding out the possible candidate plan(s), which the first agent might be executing currently.

There is a related problem of answering queries about plans. In our example, there may be a question whether *seismic survey* can be consistently done along with *enhanced recovery* under any plan. If the answer is 'yes', then the question would be what is the complete plan (or plans) under which that can be done. In the formalism described here, such questions can be answered using same pattern matching technique described in last paragraph, and it can also be done in real time, because of the availability of all plans. This may be an important advantage for an autonomous agent in any realistic environment. Isolating all possible scenarios also make it easy for updating plan data base with new information, which is a matter of adding or dropping consistent models.

In a time limited application, a time limit can be imposed when plans are generated, to create as many consistent plans as possible, rather than generating all plans. This will, of course, reduce some of the advantages discussed above. However, under some real life circumstances, that may be a better solution than to fail to generate any plan because sufficient time was not allocated.

V. Conclusion

In this article we have discussed the feasibility of planning using interval-based temporal reasoning scheme. An advantage of using temporal representation is that of incorporating uncertainty of information about the temporal relation between different temporal entities in the domain. There can be many plans feasible under this circumstances. Additional advantage of interval based representation is of higher expressiveness, which will enable new types of planning not attempted before. The problem of interval based reasoning is that of its hardness. We have devised an efficient algorithm which can find out all feasible scenarios under incomplete information, in most of the circumstances. The implication of having all feasible plans available is multifold. They are also discussed here. So far nobody has taken any look into these aspects, although practicable algorithms for finding out consistent scenario is recently coming into fore.

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):510–521, 1983.

[2] J. F. Allen and J. A. Koomen. Planning using a temporal world model. In *Proceedings of IJCAI-8*, 1983.

[3] James F. Allen, Henry A. Kautz, Richard N. Pellavin, and Josh D. Tenenberg. *Reasoning about Plans.* Morgan Kaufmann Publishers, Inc. (San Mateo, California), 1991.

[4] Mark Boddy. Classical nonlinear planning in complex domains. In *AAAI Symposium Note, the Spring Symposium on Foundations of Classical Planning*, pages 1–4, 1993.

[5] Alexander Horz. On the relation of classical and temporal planning. In *AAAI Symposium Note, the Spring Symposium on*

Foundations of Classical Planning, pages 48–52, 1993.

[6] Peter B. Ladkin and Alexander Reinefield. Effective solution of qualitative interval constraint problems. *Artficial Intelligence*, 57:105–124, 1992.

[7] Drew McDermott and Eugene Charniak. *Introduction to Artificial Intelligence*. Addison-Wesley Publishing Company, Menlo Park, California, 1985.

[8] Debasis Mitra and Rasiah Loganantharaj. Efficent exact algorithm for finding all consistent singleton labelled models. In *Proceedings of IEEE Robotics and Automation conference, Nice, France*, 1991.

[9] Debasis Mitra and Rasiah Loganantharaj. An efficient and approximate algorithm for temporal reasoning. Technical Report TR93-2-2, Center for Advanced Computer Studies, University of SW Louisiana, 1993.

[10] Debasis Mitra and Rasiah Loganantharaj. Incremental consistency algorithms for qualitative temporal reasoning. Technical Report TR93-2-5, Center for Advanced Computer Studies, University of SW Louisiana, 1993.

[11] J. Scott Penberthy and Daniel S. Weld. Temporal planning with constraints. In *AAAI Symposium Note, the Spring Symposium on Foundations of Classical Planning*, pages 112–116, 1993.

[12] Raul E. Valdes-Perez. The satisfiability of temporal constraint networks. In *Proceedings of AAAI*, 1987.

AIAA-94-1290-CP

Integrating Delibrative Planning in a Robot Architecture

Chris Elsaesser and Marc G. Slack
The MITRE Corporation
AI Technical Center
7525 Colshire Drive, McLean, Virginia 22102-3481
chris or slack @starbase.mitre.org (703) 883-6563

Abstract

Planning researchers are coming to accept that planning is not sufficient for robotics. But many of them seem to think that the concrete levels of robot control which a planner must interface are somehow uninteresting or unimportant to the development of a theory of autonomous control. On the other hand, some robotics researchers appear to think that planning is not even necessary for robotics, that reactive control will be adequate for any realistic robot system. Our thesis is that progress toward autonomous robotics will be stunted until both camps understand that the other plays an equally important role in the creation of non-trivial intelligent autonomous agents. This paper describes the role of planning and reactive control in an architecture for autonomous agents (robots). We posit this is necessary and sufficient. The key to our architecture is the interjection of a "sequencing layer" between the reactive controller needed for a robot to survive in a dynamic environment and a deliberative planner needed to develop a course of action to achieve high-level user goals.

1 Past: Robotic control as a domain for planning

Controlling an autonomous robot is mentioned in many planning research papers as a typical domain. But few planning researchers discuss, or even appear to understand the practical problems of robotic control. The following is the prototypical answer to complaints of the more practically minded robotics engineer: "I have captured the essence of the robot planning problem with a representative problem in which it is easier to present a discussion of all the important issues. The details are unimportant."

That justification would be acceptable if there were evidence that the details are really unimportant. But we know from the recent spate of papers on the complexity of planning that making planning practical are in the details. Abstractions into a "representative problem" obscures the fact that classical AI planning can only address one segment of the robot control problem.

Robotic control from the planning researcher's perspective is often viewed as the task of moving from one location to another [20, 16, 3, 15, 24, 8, 14]. The desired plan would be a sequence of movements for the robot to carry out. Such movement plans usually assume sensing systems capable of generating accurate metric models of the portion of the world relevant to the navigation task. When people got around to building robots that could benefit from this type of planning, they found that nearly all of the interesting behavior could be accomplished more efficiently by the mechanisms assumed by the planner.

The rasion d'etre of deliberative planning is the need to reason about preconditions. It is not too difficult to construct practical examples of why this is necessary for an autonomous agent. If you have more in mind for the robot than getting from here to there, you can usually convince yourself that you need to think about it before "heading in the right direction" and relying on reactivity. Dealing with complications like another agent loose in the environment or the need to do something like repair a broken device (so you need to gather the right tools before you leave) are not activities a purely reactive system can accomplish any more efficiently than a classical planner can accomplish movement planning in unstructured environments.

Chapman proved that planning is computationally intractable and could not possibly be a sufficient theory of intelligent behavior in a real-time environment [5]. Others have catalogued some of the behaviors one would need in an autonomous agent: reaction to unforeseen events, iterative actions (e.g., traveling down a street stopping for all the red lights), real time projection and conditional commitment to action (e.g., cross a busy highway [19]).

Copyright © 1993 American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

Chapman was the vanguard of the "situated reasoning" approach to intelligent agents [4, 1, 6, 17]. These "reactionaries" started at the opposite end of the control continuum from classical AI planning, bent on showing that planning may not even be necessary for intelligent behavior. But the problems the reactionaries cannot address turned out to be important to producing an autonomous agent. Among the more obvious are the inability to employ predictive reasoning about preconditions (check the gas gauge before embarking on a long car trip, don't wait until the car starts sputtering), coordinating activity with other intelligent agents (pick up a heavy object together), reason under uncertainty and take risk-alleviating actions (move into the right lane well before your exit).

Our hypothesis is that planning is necessary for control of autonomous robots, just as situated reasoning or reactive behavior is necessary. The main issues are mediating between deliberation and reaction to produce seamless intelligent behavior, and determining the proper roles of the various components of an intelligent agent architecture.

2 A three-layer architecture for intelligent agents

The robot intelligence community has begun to agree on a software architecture for "intelligent" robotic systems [10, 18, 12, 7, 11, 23, 25]. The consensus emerging is an architecture which incorporates both planning and reaction. In most of the architectures cited, there is a planning component for reasoning about the overall mission and generating contingencies when the mission itself is in danger of failing. Importantly, the planner is asynchronous (but on-line) with a real time reaction component which can achieve the situated reasoning needed for survival and continuous control.

While it now seems obvious there is a role for reaction and planning in robot control, what is not so obvious is how to mediate between the two. Our architecture separates the general robot intelligence problem into three interacting pieces, with the middle piece being the key to mediation between reaction and deliberation (see Figure 1):

1. A set of robotic specific reactive skills. For example, grasping, object tracking, and local navigation. These are tightly bound to the specific hardware of the robot and must interact with the world in real-time.

2. A sequencing capability which can differentially activate the reactive skills in order to direct changes in the state of the world and accomplish specific tasks. For example, exiting a room might be orchestrated through the use of reactive skills for door tracking, local navigation, grasping, and pulling.

3. A deliberative planning capability to reason in depth about goals, preconditions, resources, and timing constraints. The planner generates rough plans for accomplishing goals. For example, given the task to retrieve an item and a map of a building, the deliberative system could reason about the interconnection of spaces and return a plan for the robot to exit the room, follow the hall to the left and enter the third door on the right.

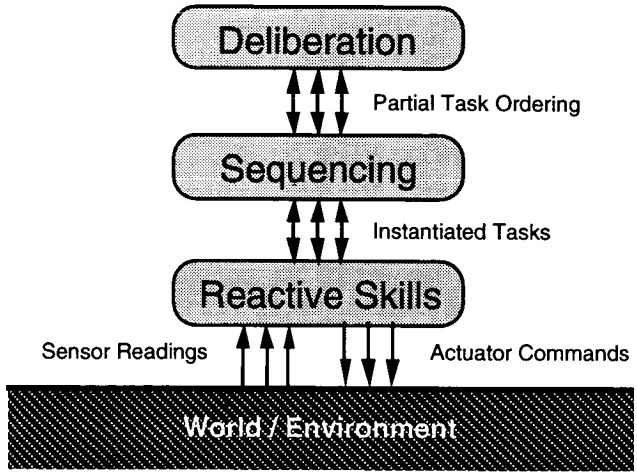


Figure 1: Generic intelligent control architecture

This paper focuses on the interaction of planning and sequencing layers of this architecture. Interaction of the sequencing system with the reactive layer of the architecture is covered in other papers [23, 25].

3 Sequencing: caching techniques for handling routine activities

Reactive control addresses only the most obvious defect of state-based planning for robots, the inability to represent and reason about motor control in real time. But even if provided with primitives such as grasp, track-wall, and so on, a state-based planner will still be unable to efficiently handle everything needed for robot control above the level of reactions that can be compiled into guaranteed-reaction-time primitives.

The most obvious problem is the need to do indefinite iteration of sequences of primitive behaviors:

```
Do unitl (at robot1 door5):
Put-one-foot-in-front-of-the-other
```

The problem with such sequences for a state-based planner is they produce an indeterminate number of states to manage. A plan is basically a proof that some goal can be achieved with a sequence of state transitions effected by atomic plan steps. Planners convince themselves a goal is achievable by constructing the whole plan. Obviously, it is difficult to generate a complete state-transitions plan to embody what we easily expressed by the iterative construct above. That is the reason typical planning representations ground out on primitives like the following:

```
(Operator move
   :purpose (location ?robot ?destination)
   :preconditions ((clearpath ?robot ?destination)
                   ...))
```

Such operators assume things which are hard to encode as states, like keeping away from walls and people. They also assume indefinite sequences of very small grain actions. Situated actions typically implemented in robots often match the state transition view of classical planning perfectly well [13]. The problem is two-fold. One is performance; at the level of abstraction provided by situated actions there would be too many plan steps to generate for even very simple goals like moving from within a room out into a hallway. The second is representation. No one has developed a useful state-based planner that can reason about indefinite iteration of actions and conditional actions.

What is needed is a layer between reactive behaviors and deliberative planning which allows the planner to reason only to the level of routine activities such as move and open-door (grasp, turn, and pull. If that fails, push. If that fails consider another route to the goal). In our architecture, we use Reactive Action Packets (RAPs) to encode routine behavior as a sequence of situated skills [22]. RAPs is a language with a syntax similar to the syntax of classical planning systems [10]. Like most planning systems, the RAP system uses a library of decomposition rules to represent sequences of behaviors to accomplish a task. The system can quickly transform a task into a context specific sequence of primitive actions by caching solutions to common tasks. Unlike a planning system's computationally expensive search mechanisms used to decompose tasks into primitives, the RAP system must have a solution to the given task cached in its library or the system reports a failure. RAPs can encode conditional and iterative sequences of actions since there is no state-based search involved. As is exemplified by the following example, the door is iteratively bashed with a sledge hammer until the not closed state is detected or simply opened if the door is unlocked.

```
(define-rap opendoor
(success (not (closed ?currentdoor)))
 (method
     (context (doorlocked ?currentdoor))
     (t1 (grasp-sledge-hammer) (for t2))
     (t2 (pound-door ?currentdoor)
          (wait-for (not (closed ?currentdoor)))))
  (method
      (context (not (doorlocked ?currentdoor)))
      (t1 (grasp-door-handle) (for t2))
      (t2 (turn-knob) (for t3))
      (t3 (pull-open-door))))
```

While the RAP system can perform task decomposition, it is not suited for direct interaction with the world. The software constructs that are used for selecting action routines, binding variables, and so on make the system too slow for survival. Thus the system is used in our architecture to dynamically configure a reactive layer to handle the interaction with the world for the current task and situation. This allows complex behaviors to be programmed, while relying on always-active situated skills to protect the robot from inaction in a rapidly changing environment.

Sequencing, married to reaction, yields significantly better task coverage than either of the two can provide alone. Still, the combination of the sequencing and reactive layers is not structured to perform complicated resource allocation reasoning. Such is typical in determining the best way to carry out a set of tasks. Nor are these two layers good at reasoning about the failure requirements or consequences of a task. So where the sequencer gains in its ability to handle routine situations (e.g., starting a car, opening and moving through a door), it lacks the ability to string these routine tasks together in a way that will have the desired "global" behavior. But that happens to be just the thing planners are good for.

4 Planning

Our view is that there is a role for state-based planning in robotic intelligence, but it should be limited to tasks that are not easy to specify as sequences of

common robotic skills. When planning is necessary, the planner should think of the problem at the highest level possible in order to make the problem space the smallest possible.

Thus, the role of planning is to deliberate, but only when necessary. The role of reaction is to control real-time behavior. The role of sequencing is to raise the level of abstraction of the lowest level of activities which the planner will concern itself. In the process, eliminate the need for state-based planning of things which are easy to encapsulate in an operator that grounds in an indefinite iteration of low-level skills. Importantly, all three layers must operate concurrently and asynchronously. Accomplishing this is the key to making planning useful in a robot.

Because the sequencer has a cached solution to routine tasks, the planning system has the advantage of building upon this level of abstraction providing it larger grain sized primitives. This eases the complexity of the "planning problem" because it eliminates large numbers of essentially linear planning problems. Ability of the RAP system to deal with iterative behavior greatly simplifies the planner's representation, allowing the simple state representation common to classical planning to suffice in most common situations.

The planner we are using in our experiments is called AP [9]. AP has a number of features which make its role more compelling than robot planning typically exemplified in the planning literature.

One thing missed by both the planning and robot control communities (while they were arguing over the necessity of planning) was autonomous robots generally will be autonomous only from the human giving them orders. They will not often be acting alone in their environment. Multiagent control is necessary when more than one robot is employed to carry out tasks, or when multiple robots are operating independently on multiple tasks in a shared environment.

AP was designed to deal with multiagent coordination. To do this, it extends state-based planning to reason about the conditions that hold during actions. This capability allows AP to plan activities such as two robots carrying a bulky object. The following operator is an example from a test domain. Note the planner can instantiate the variables ?arm-or-robot1 and ?arm-or-robot2 with anything that meets the constraints. A two-armed robot or two single armed robots might be used. The temporal relation "simultaneous" imposes a non-codesignation constraint on the agents so that a very strong one-armed robot would not qualify. Other temporal constraints in the plot language would allow codesignation). These plot temporal constraints also cause the plans steps that instantiate the plot subgoals to include scheduling information that the sequencing layer and AP's execution monitor can use.

```
(Operator pickup-heavy-object
 :purpose (holding ?planner ?large-thing)
 :arguments
    ((?weight-of-thing
       (get-value ?large-thing 'weight)))
  :preconditions
     ((top ?large-thing clear)
      (on ?large-thing ?something))
  :constraints
    ((can-lift ?arm-or-robot1
               (* 0.5 ?weight-of-thing))
     (can-lift ?arm-or-robot2
              (* 0.5 ?weight-of-thing))
  :plot
    (simultaneous
      (grip ?arm-or-robot1 ?large-thing)
      (grip ?arm-or-robot2 ?large-thing))
 :effects
    ((holding ?planner ?large-thing)
     (top ?something clear)
     (on ?large-thing nothing)))
```

Another feature of AP which makes it appropriate for control of autonomous agents is that it can reason about uncontrolled agents. AP was originally developed to address multiagent adversarial domains (AP stands for Adversarial Planner). Of course most robot applications are not adversarial. An uncontrolled agent might be a human operating in the environment along with a robot, or even nature. AP can use its adversarial reasoning capabilities as a risk assessment mechanism to decrease the probability of dangerous interactions with other agents.

AP includes a "counterplanning" component to reason about how an uncontrolled agent might prevent a plan from succeeding, either by negating a precondition or a "during condition." Problems are uncovered and addressed by augmenting the plan with operations which prevent the negative effects of the uncontrolled action. This amounts to reasoning about situation-specific preconditions, and is the way AP addressees the "qualification problem" [21].

A typical example of this type of reasoning in a robot application might go as follows. The robot is assigned the task of repairing a device. In the course of the planning process it might post a "protection interval" on the condition that a door remain open for the duration of some operation. Counterplanning

might discover that a human could close the door, and a way to prevent this could be to post a notice that the door must be open until further notice.

AP has other features which are needed to fully address the requirements of an intelligent agent. These include: reasoning about metric time (scheduling), execution monitoring, and replanning. Execution monitoring allows the agent using AP to recognize and skip plan steps that are overcome by events, and to replan when something unexpected occurs (e.g., a door that should have been left open is closed and locked). Execution monitoring also projects ground truth observations through the state space generated during planning. When ever there is a change in a output situation proposition, AP's execution monitor rechecks preconditions and recalculates certain "recomputable effects" of subsequent plan steps. This permits AP to predict failures, rather than waiting to notice them, as would happen if the agent relied only on situated reasoning and sequencing. This is obviously something the planner should be doing outside the sense-act cycle of the plan executor. In fact, AP was designed from the beginning to be a deliberative process aloof from the execution environment.

Replanning in AP is cued by the execution monitor when it notices or predicts failures. Replanning in AP is based on the concept of a "minimal repair wedge." AP assumes the majority of a plan is salvageable. The idea is to excise the minimum number of plan steps dependent on the failed step and replace them with a "wedge" of operations that achieve the original subgoal with alternate means. This strategy is both more computationally efficient and cognitively plausible than planning again from scratch, which is what most classical planners are doomed to do.

5 Implementation Status

We have completed implementation of an interface between a RAP-based sequencing system and a facility for providing a set of situated skills which the sequencer can manipulate to cause activity in the world [22, 25]. We are now combining AP with the RAP-based sequencer.

Shortly we expect to begin testing our architecture on a number of problems. Questions we plan to address include the following:

1. What activity is appropriate for planning or sequencing layers?
2. What domains where reactivity is sufficient?
3. Are there domains where sequencing is sufficient?
4. In which domains is deliberation highly useful?
5. To what measure is the architecture beneficial over more ad-hoc approaches?

6 Future Work

After we complete testing our architecture, we plan to explore the architecture as a basis to incorporate in robots certain cognitive capabilities normally associated with intelligent behavior. The first area we will investigate is learning.

We are guided in our ideas about learning by Anderson's ACT* model of cognition [2]. In ACT*, one of the important uses of learning is for performance improvement. It is hypothesized that expertise is gained by compiling common sequences of primitive actions into routines which henceforth take the agent essentially no time to derive. This type of learning can be easy in our implementation because the RAP library can be dynamically modified. For example, the robot could "learn" the commonly used plans for getting places from its nominal home (e.g., to the mess hall, the latrine). Since AP's plans are parameterized and have syntax similar to RAPs, the system could compress common sequences of operators into RAPs and then add that RAPs as a planning primitive. Thus, the next time the robot has a goal to eat dinner it would not have to invoke the planning system to accomplish the task of getting to the proper location.

References

[1] P. E. Agre and D. Chapman. Pengi - an implementation of a theory of activity. In *Proceedings of American Association of Artificial Intelligence Conference*, pages 268–272, July 1987.

[2] J. R. Anderson. *The Architecture of Cognition.* Harvard University Press, 1983.

[3] R. A. Brooks. Solving the find path problem by a good representation of free space. In *Proceedings of American Association of Artificial Intelligence Conference*, pages 381–386, 1982.

[4] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[5] D. Chapman. Planning for conjunctive goals. Artificial Intelligence Laboratory 802, Massachusetts Institute of Technology, 1985.

[6] J. H. Connell. Creature design with the subsumption architecture. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1987.

[7] J. H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, April 1992.

[8] J. G. de Lamadrid. Obstacle avoidance heuristic for three dimensional moving objects. Computer Science 87-16, Institute of Technology at the University of Minnesota, February 1987.

[9] C. Elsaesser and T. R. MacMillan. Representation and algorithms for multiagent adversarial planning. Department W153 MTR-91W000207, MITRE Corporation, December 1991.

[10] R. J. Firby. *Adaptive Execution in Complex Dynamic Worlds.* PhD thesis, Yale University Department of Computer Science, January 1989. see Technical Report 672.

[11] R. J. Firby. Building symbolic primitives with continuous control routines. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems.* Morgan Kaufmann, June 1992.

[12] E. Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots.* PhD thesis, Virginia Polytechnic Institute Department of Computer Science, April 1991.

[13] L. P. Kaelbling. Goals as parallel program specifications. In *Proceedings of American Association of Artificial Intelligence Conference*, 1988.

[14] L. Kavraki. Computation of configuration-space obstacles using the fast fourier transform. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1993.

[15] J. P. Laumond. Model structuring and concept recognition: Two aspects of learning for a mobile robot. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 839–841, August 1983.

[16] T. Lozano-Perez and M. A. Wesley. An algorithm for planning collision free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[17] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. Technical Report 1228, Massachusetts Institute of Technology, May 1990.

[18] D. P. Miller. Rover navigation through behavior modification. In *Proceedings of the NASA Conference on Spacecraft Operations Automation and Robotics*, July 1990.

[19] J. C. Sanborn and Hendler J. A. A model of reaction for planning in dynamic environments. *International Journal for Artificial Intelligence in Engineering*, 3(2):95–102, 1982.

[20] M. I. Shamos. *Computational Geometry.* PhD thesis, Yale University Department of Computer Science, 1975.

[21] Y. Shoham. *Reasoning about Change.* The MIT Press, 1988.

[22] M. G. Slack. Computation limited sonar-based local navigation. In *AAAI Spring Symposium on Selective Perception*, pages 142–146, April 1992.

[23] M. G. Slack. Sequencing formally defined reactions for robotic activity: Integrating RAPS and GAPPS. In *Proceedings of the SPIE Conference on Sensor Fusion*, November 1992.

[24] C. E. Thorpe. Path relaxation: Path planning for a mobile robot. In *Proceedings of American Association of Artificial Intelligence Conference*, pages 318–321, 1984.

[25] S. Yu, M. G. Slack, and D. P. Miller. A streamlined software environment for situated skills. In *Proceedings of the AAIA/NASA Conference on Intelligent Robots in Field, Factory, Service and Space*, March 1994.

AIAA-94-1291-CP

PLANNING IN SUBSUMPTION ARCHITECTURES

Eugene C. Chalfant

University of Southern California
Los Angeles, California
echalfan@pollux.usc.edu

Abstract—*A robotic planning and control system based on the subsumption architecture is described. The subsumption planner extends the purely reactive subsumption architecture by extending the sensor space (from which the behavior modules are triggered) into a virtual future, and augmenting the behavior module with a cause-effect predictor triggered by the same sensory situation as the reactor. Virtual sensor space is used by the planner as a scratchpad to visualize alternate plans. The predictor contains a partial world model relevant to its particular behavioral expertise. The collective network of predictors operates in parallel with the reactive network forming a recurrent network which generates plans as a hierarchy. Details of a plan segment are generated only when its execution is imminent according to the principle of least commitment. An implementation of subsumption using object oriented design is proposed. The behavior of the subsumption planner is demonstrated in a simple maze navigation example. The subsumption planner is expected to improve the robot's performance by reducing feedback delays and unnecessary detours. It provides a framework for general behavioral planning in real-world robots of the subsumption style.*

1. Introduction

Reactive robotic control systems, such as the subsumption architecture, have enjoyed popularity among the research community for the last few years. Robots built according to these principles are very successful at performing tasks in unstructured, real-world environments. However, reactive systems tend to behave in a pre-programmed, rote manner. Selecting alternate courses of action based on previous experience or reasoning (i.e., deliberative behavior) must be designed into the behavior itself rather than being an emergent property of the network interconnectivity. Recent work has recognized a need to incorporate deliberative planning capabilities in real-world systems.

Planning is essentially the ability to look ahead and predict outcomes of actions (or inactions), and to make decisions based on that knowledge. A plan is a sequence of decisions to be made in the future. These decisions assume a particular expected sequence of states. Anticipation of future states gives the planning robot advantages in efficiency (via the ability to avoid known dead ends), in flexibility (replanning on the fly), and in reaction speed (eliminating most of the delay inherent to reactive feedback-only systems). Autonomy in the real world demands predictive and deliberative behavior in addition to a reactive component.

The subsumption planner uses a parallel distributed computational paradigm based on the subsumption architecture for the control of real-world-capable robots. This approach is derived from a number of various disciplines including ethology, the theory of animats, and computational neuroscience. Plans are represented as trajectories in a time-extended virtual sensor state space. States along this trajectory represent the future as the robot expects it to appear, in terms of its own senses. Virtual sensor state space is used as a planning tool to visualize the robot's anticipated effect on its environment.

Decision sequences are generated by the planner based on the environmental situation expected at the time the robot must commit to the decision by acting on it. Between these decision points, the robot performs in a pre-programmed manner, limiting its reactions to avoiding obstacles, stalls, and danger. A rudimentary, domain-specific partial world model contains enough information to extrapolate the end results of rote behavior between decision points, and thus to predict the sensed situation at the next decision point.

By constructing plans with little detail as long as they are far in the future and filling in details only as their execution becomes imminent (the principle of least commitment), failed plans can be discarded without much resource cost. Rough plans are represented as indefinite trajectories between a few well-defined waypoints in the state space. These waypoints are the expectations generated by the predictors. Initially, they are spread far apart in time—the plan is coarsely resolved. Rough planning consumes few resources. As the first plan segment comes closer to fruition, details are added

Copyright © 1993 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

just in time for execution. These multi-resolution plans are built from the outside in.

Planners need a world model to evaluate alternatives. One objective of the subsumption planner is to distribute the world model in a plausible way across the entire decision-making system. Each behavior module contains a discrete piece of expertise. These expert agents contain not only action generation routines, but also cause-effect predictors. The planner is implemented as a network structure which parallels the behavior module network of the subsumption architecture.

Implementation of subsumption as an object-oriented design simplifies the implementation of a software simulation, as well as providing organizational and developmental flexibility. The subsumption network structure lends itself to the use of distributed hierarchical encapsulating objects. In addition, the hierarchy enables prioritization of active behaviors to occur before the generation of actual motor outputs. This simplifies the prediction mechanism.

2. Related Work

As the shortcomings of monolithic, centralized robotic controllers and planners became obvious, new distributed architectures were proposed. These were rooted in a variety of different nontraditional disciplines, including ethology, biology, neuroscience, physics, psychology, and sociology. A strongly ethological and intuitively elegant approach is espoused by Albus[1]. Marvin Minsky, a pioneer in artificial intelligence, theorizes about the nature of information processing in the human mind, speculating that thought is composed of the collective actions of a society of individual, subintelligent agents acting cooperatively[2]. The new architectures share many features with object-oriented paradigms. Schema theory[3], in particular, models intelligent systems from the perspective of brain theory as hierarchical networks of nested object-like schemas. Schema theory and object-oriented design are both rooted firmly in distributed artificial intelligence.

Sensor and motor information (indeed, any type of information at all) can be described in terms of dynamic high- or infinite-dimensional state spaces[4]. A trajectory through this state space represents the time evolution of information or concepts. This generic framework has been developed into a description of physical and mental behavior, which can describe many diverse information processing techniques and knowledge representations.

The subsumption architecture developed by Brooks at MIT[5, 6, 7] introduced a new perspective on building robots for the real-world, often called creatures or animats. Connell[8] further develops the subsumption technique, describing an autonomous robot whose job it is to find and collect soda cans in a lab and deposit them in a receptacle. This robot has served as a testbed for an in-depth study of subsumption and its numerous advantages over earlier methods for controlling real-world robots[9]. The major achievement of the subsumption architecture is to demonstrate how to combine many isolated pieces of robot control into a single, working real-world machine operating under a single, coherent, and consistent architecture.

Earlier robots were effective only in toy worlds: well-defined environments, such as in manufacturing, where repetitive movements could be guaranteed to be successful. These applications do not require much sensory ability. The robots built by Brooks react in real-time to changes in the environment. They rely on a rich suite of sensors to provide information about the environment on which to act. The actions themselves are generated by a network of behavioral modules whose outputs are mediated by a hardwired arbitration network of gating nodes. This network gives certain higher-level behavioral modules responsibility for, and control over, the outputs from lower-level modules.

In addition to performance advantages due to direct implementation on parallel hardware, the subsumption architecture provides all the benefits of a distributed system, including scalability, graceful degradation, robustness, simpler elements, and biological plausibility. Much of the intelligence of the system emerges from, and is embodied in, the network connectivity.

The lack of an explicit world model was originally seen as an advantage of subsumption. Using the real world as its own model solved the problems of keeping the model up to date and deciding what was important to include, as well as what form of representation to use. However, without a world model, a system or organism cannot anticipate the effect of an action until that effect is actually sensed (giving rise to feedback delays). This can lead to poor reaction time and potentially life-threatening slowness.

The purpose of a world model is twofold: It keeps track of the state of the world, and it describes cause-effect relationships due to actions (i.e., moving an object) or inactions (allowing an object to fall). These are data structures and processes re-

spectively. The cause-effect relations may be due to laws of physics, to the robot's own actions, or to the actions of another agent, in order of increasing difficulty of prediction. In the subsumption architecture, these cause-effect relations are implicitly designed into the behavior modules. For example, termination conditions assume a cause-effect relationship due to physical laws. The result of actively moving an object from x to y is that the object eventually exists at y. The fact that the object is at y can either be represented internally in a world model, or, in the subsumption model, sensed again only when necessary.

Recently, several modifications and extensions to the subsumption architecture have been proposed. These address limitations or inefficiencies of subsumption such as the lack of a world model, or its lack of learning ability.

Mataric[10] extends subsumption by implementing knowledge representation as a map (a kind of world model) for navigation by integrating an internal landmark-based map representation built as behavior modules. Behavior modules are used here to *represent* as well as *act*. In this case, each module represents a landmark. The building blocks of the subsumption architecture remain fundamentally the same; Mataric describes a new usage of them.

Dorigo and Schnepf[11] propose a learning technique for behavior-based robots in which new behaviors are created and added to the robot's repertoire using a genetic algorithm. The behavior-based architecture is based on ethological theories developed over the last eighty years[12]. They note that behaviors have been constructed which are well tailored to specific tasks, but no conceptual model exists for relating behaviors to each other. Learning is achieved by evaluating new behaviors and including them if they perform well. The resulting system is a synthesis of behavioral and genetics-based paradigms for intelligent real-world behavior.

Learning by progressive modification of a repertoire of reactive behaviors is described by Lyons[13], using a process-algebra language. The concept of a planner as a separate subsystem which interacts with a reactor is presented. The plan is contained in the reactor as a set of processes. The reactor adds and deletes processes, tuning the reactor to perform the task more robustly over time. The planner has a repertoire of plan elements and the knowledge of how to fix shortcomings in the reactors overall behavior. It provides the system with a global perspective on task execution which is lacking in the purely reactive system.

Traditional planning systems constructed plans off-line, often in a separate centralized planning subsystem. Planners were seen as logical engines which generated a complete, detailed plan as a result of a search through a tree of possible action sequences[14, 15]. These plans worked best in a toy world. Distributed planning systems were eventually developed as a consequence of the focus on distributed artificial intelligence in the late 1980's. One of these was the Distributed Vehicle Monitoring Testbed[16, 17, 18] which distributed partial plans among independent agents. In this scheme, plans were subdivided spatially.

In the past, planning has been studied in isolation. It is now becoming widely accepted that the predictive, deliberative planning component is necessary to complement the reactive behavior-based system. These two systems must be highly integrated, yet distributed across the network. Planning, in a sense, represents the antithesis of reality-based reactive systems. A planner must imagine nonexistent realities and visualize and evaluate alternative future actions in that virtual reality.

3. Architecture

Reactive robot controllers implement a feedback loop for all activity (Fig. 1a). Feed*forward* loops (Fig. 1b) are more responsive and exhibit smoother, more desirable control, provided the model is accurate. The feedforward control technique has been used to mimic the smooth, precisely controlled behavior of the human arm by a robot arm using highly non-linear air-bladder "muscles" [19, 20, 21]. The feedforward loop relies on an internal model of physical cause and effect to predict and generate appropriate actions in real time, before their effect can be sensed. The Kawato-Katayama arm trajectory generators, implemented as neural networks, act as continuous-time predictive models the physics of arm behavior.

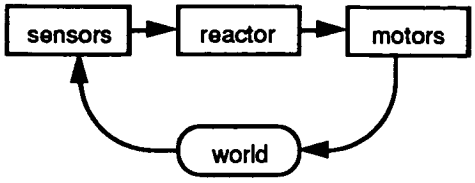


Fig. 1a. Feedback only reactive system

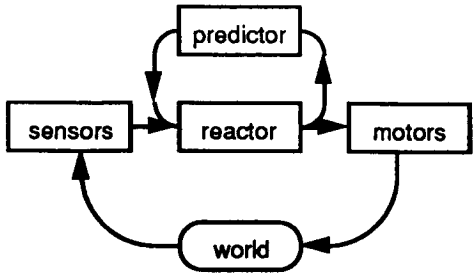


Fig. 1b. Feedforward reactive/predictive system

The subsumption planner adds to traditional subsumption the ability to predict the effect of robot actions. The action-generation expertise is collocated with the knowledge of cause-effect relationships in the behavior module associated with that domain. This cause-effect knowledge is encapsulated in a predictor which anticipates the expected state resulting from performing the behavior. The expectations are represented as state vectors in the virtual sensor state space representing future states—the collection of these states together define a trajectory representing a future course of action.

Subsumption networks are often seen in which information flows in one direction from input to output. There are no recurrent links in this type of network. The lack of recurrent links allows only simple reactive behavior to be produced, precluding complex dynamic behavior. In the same way, non-recurrent neural networks such as the back propagation network do not contain any recurrent information flow. Non-recurrent networks are not dynamical systems, and therefore they cannot exhibit chaotic or even oscillatory time-varying behavior. This severely limits the behavioral complexity and sophistication of these networks (although they are more easily understood). Similarly, in the subsumption architecture, any complex processing occurs *within* the behavior modules, rather than between them. (Some specialized subsumption networks have been designed with recurrent links; those for the control of walking, for example.)

The subsumption planner allows for three general forms of information flow. The sensor subsystem contains an afferent abstracting flow, and the motor subsystem contains a de-abstracting efferent flow. The subsystem between these two, comprising the decision making mechanism, is constructed as a network which allows a great number of information flows in both forward and backward directions. The forward flows are essentially identical to those of the conventional subsumption architecture and represent reactive behavior such as tracking or avoidance. The backward flows create loops in the information flow; these recurrent flows generate the visualization of future actions and planning. These recurrent flows are model generated predictions which are fed *forward* (i.e., in the same direction as information in the real world—from effectors to sensors) toward the sensory subsystem and treated as imaginary, virtual sensory situations. The key point is that the internal modeled information flow mimics the outside world's physical cause-effect information flow. Multiple iterations of this loop propagate predictions farther into the future, although any modeling errors will accumulate. These sensory situations are ordered in time and represent plans for future behavior.

In the traditional subsumption architecture, the responsibility for reacting to the environment is decomposed and delegated to a number of reactive behavior modules. A behavior module can be seen as an agent which is an expert in one discrete domain of behavior. These modules in turn may have hardwired managerial control over an entire network of subordinate behavior modules. The control is manifested as subsumption, whereby the manager commandeers control over the robot whenever it wants. The manager handles special situations only; the common situations are ignored by the manager and handled by its subordinates. If the specific triggering situation for the manager does not exist, some subordinate, more generalized behavior probably *has* been triggered by a less specific sensory situation. At the lowest level, a behavior may be continuously triggered by default; in effect, it is triggered by *any* sensory situation.

4. Extensions to Subsumption

In order to implement the subsumption planner, four extensions to the traditional subsumption architecture are needed:

- Sensor and motor space abstraction
- Virtual future sensor space
- Cause-effect predictors: augmentation of behavior modules

- Hierarchical organization: object-oriented implementation of arbitration

Sensor and motor space abstractions extend the real-world interface (input/output) information spaces, allowing for more sophisticated, abstract triggering and action-generating mechanisms for the behavior modules. Virtual future sensor space provides a scratchpad for construction of plans. Cause-effect predictors are associated with the behavior modules and generate the plans themselves as trajectories in the virtual sensor space. The hierarchical organization is an alternate object-oriented implementation of the subsumption architecture.

4.1 Sensor and Motor Space Abstraction

Sensor space is a very-high-dimensionality representation of the information entering the robot from its external sensors. After processing and combining raw sensor data, abstract sensor data containing composite information in a more easily assimilated form, or a form with higher information density, can be created. These abstractions have been called "logical sensors"[22]. They are generated through the fusion of raw sensor data. Examples in robot vision include edge detectors, novelty detectors, motion detectors, or highly abstract sensors such as face recognizers. Cross-modal abstractions sense location or orientation using a number of sensor sources to reduce uncertainty. A logical sensor might determine location by combining information gleaned from a number of raw sensors, such as sonar, visual clues, radio landmarks, etc.

In the original subsumption architecture, behaviors are triggered directly from raw sensor inputs. The entire extended sensor space including raw sensor data and abstract, derived sensor data, is available to trigger the subsumption planner's behavior modules. Any particular behavior module is sensitive only to a small localized subset of this sensor space, similar in some measure, for example a small visual patch. The sensitivity of a behavior module is also localized in level of abstraction—a simple, low-level behavior may trigger on contact with a single touch sensor; a more abstract, higher-level managerial behavior on recognition of a complex object such as a familiar face.

The abstract sensors are derived from the raw sensors; the raw sensor space by itself is complete. The abstract sensor space extension is a conceptual device to provide behavior modules with a common pool of sensed information from which to trigger. A behavior module, rather than triggering on a complex pattern which could potentially be multi-sourced and multi-modal (as well as uncertain and noisy) in raw sensory space, may trigger on a simple abstract pattern, or even a single highly processed attribute in abstract space.

These high dimensional spaces are conceptual constructs. Obviously, to implement them faithfully following the theory is extremely wasteful and probably impossible. The implementation can be drastically optimized while still following theory. For example, to conserve resources, the abstraction of raw sensors and data fusion should only be performed by request.

Complex motor actions generated by the motor subsystem are also abstracted into motor pattern generators (called "central pattern generators" by neuroscientists). Locomotion is a repetitive abstract motor pattern which is comprised of a number of individual motor actions. The relatively complex repetitive pattern of actions are generated by a single manager behavior which directs simpler, lower-level behaviors to generate the rudimentary component actions. Variations of locomotion, such as speed, gait, direction, can be supplied to the abstract locomotion behavior as parameters.

4.2 Virtual Future Sensor Space

Sensor space, extended in dimensionality by the incorporation of abstract sensor dimensions, is also extended orthogonally to represent a time dimension—the extension of the sensed environment into the future. This extension will be used to provide a working scratchpad for plan generation.

This extension provides a conceptual framework for the construction of plans. A plan is a visualization of a sequence of events. The most complete, natural, and efficient way (for that matter, the only possible way) to represent a visualization is in terms of the same modality in which it will actually be sensed. Since the extended sensor space contains all possible sensory situations, the visualization of "how it will appear to the senses" is also contained. The difference between real-time sensory situations and virtual sensory situations is in completeness (everything need not be represented), resolution (irrelevant details may be omitted), and uncertainty (multiple possible values or fuzziness).

The plans generated by the predictors are represented as a sequence of nodes or waypoints, relatively firmly located in state space, at distinct future times, connected by indefinite links. As the plans mature or become more imminent (the plans are also refined by the predictors), they become

more firm in location, and intermediate waypoints emerge.

Multiple alternative plans are tagged with plan quality measures. The plans are evaluated by the predictor. When a choice of plan is necessary (i.e., when a complete replan is necessary, or upon embarking on a new task), the highest quality plan is selected. The robot will then begin executing this plan.

4.3 Cause-Effect Predictors

Predictors are agents associated with behavior modules. The original subsumption style action-generating behavior module, which we now call a *reactor*, is triggered by a sensory situation in the current sensor subspace. By augmenting the behavior module with a predictive mechanism, called the *predictor*, the effect of the reactor on the environment can be predicted. The predictor is triggered by the same sensory situation as the reactor in the current *or any future* sensory subspace. The predictor then generates a trajectory from the trigger point to the future point representing the predicted resultant effect of performing the behavior.

Predictors are not required for low-level, reflexive, or protective behaviors such as obstacle avoidance or tracking. These behaviors avoid anomalous states, such as being stuck in a corner; they guarantee that the robot remains in a nominal space. Higher level behaviors can expect that the anomalous states will be avoided. The lowest level behaviors are purely reactive. This reliance on low-level behavioral guarantees make the prediction job of the higher level behaviors easier. In general, predictors are necessary only for manager behaviors.

The predictor is insensitive to its trigger source—whether it is actually being sensed at the present moment, or if the sensory situation is an imaginary construction of some previously triggered predictor. The idea is to visualize a sequence of events using exactly the same sensory computational pathway as would be stimulated during the actual performance of the sequence. The differences are that: 1) the actual raw sensor transducers are not stimulated, 2) *conceptually*, the behaviors are displaced along the time dimension of the sensor space, and 3) the reactors do not generate motor commands. The same mechanisms used for reactive behavior (the trigger mechanisms and the arbitration of actions) are used for planning.

Obviously, the prediction is defined only in the subspace in which the reactor may potentially have a repeatable, predictable effect—a reactor which closes the robot's gripper will, in general, have no predictable, correlatable effect on sensed ambient light intensity.

As in schema theory, a subsumption planner predictor is a black box. The internal mechanism of the predictor is not important to the functioning of the entire distributed system; only the externally observable functional behavior (i.e., its role in the architecture) must be well defined. The implementation may use a procedural algorithm, a neural network, specialized hardware, or any other technique to perform its function.

4.4 Hierarchical Organization

While planning, no outputs should be generated from the activity of the behavior modules. The reactor does not generate output if the trigger is in the virtual sensor space. However, since the subsumption architecture's arbitration network (the collection of output wires along with their connectors) are connected to reactors which generate outputs whenever they are triggered, it is difficult to determine the actual controlling behavior module without actually generating motor outputs to send through the network. This would require some sort of action gating mechanism at the output of the arbitration network to inhibit the passage of motor commands. The result of the arbitration needs to find its way back to the predictor also.

The connectivity of the subsumption network defines its global behavior. Behavioral modules which are triggered upon sensing special environmental situations subsume lower level behaviors (if the subsuming behavior was not triggered by a special case of the subsumed behavior, the subsumed behavior would not be triggered in the first place, and the subsumption relationship would be inappropriate, i.e., in a properly designed network the subsumed behavior should already be triggered and operating when the subsuming behavior takes over). In a sense, the manager behavior commandeers control of the entire system because it believes it is best suited (in the eyes of the designer) to deal with the problem at hand.

The triggering of a module does not automatically cause the robot to perform the actions generated by this module. Several behavior modules may be triggered by the same sensory state. The actions must be mediated by the subsumption or arbitration network. This network determines which behavior may assume control of the entire system at any one time. In general, behaviors which are triggered by a smaller, more exclusive region within the sensory space will be regarded as more appropriate than more generic behaviors. The result is

that behaviors which trigger upon special situations, or exceptions to the general rule, will have priority over other behaviors. These more sophisticated, specialized manager behaviors "subsume" lower-level, or simpler behaviors.

In order to retain "ownership" of the motor commands by the generating behavior, an alternative is to perform the arbitration before the commands are output by the reactors. By doing this, the behavior module knows that it has been selected (on the basis of either real or virtual senses) as the controller of the entire system, and the predictor can act accordingly. If the trigger source is the virtual sensor space, the arbitration still occurs, and the result is known locally to the behavior module.

The subsumption architecture triggers all eligible (even inappropriate) behavior modules to generate actions in parallel. By giving each module some awareness of its own place in the control hierarchy, and some visibility into the activity of behaviors which may override its own eligibility, the number of eligible, active modules may be reduced to those which may actually control the robot. (Note that several subsumption networks may control a robot, so that several different behavior modules may be generating actions at the same time for different subsystems). This type of hierarchical selection lends itself to the use of schema theory and object-oriented software techniques.

The arbitration network composed of suppression, inhibition, and default nodes in the subsumption architecture functions similarly to the schema assemblage construct in schema theory. A schema assemblage is a collection of (possibly interconnected) schemas and is itself regarded as yet another schema. The component schemas are definitions rather than instantiations, so that multiple inclusions in different schemas are realizable. This self-similar hierarchical structure lends itself to object-oriented software construction techniques. Object classes are built from less specific classes as specializations of those classes in the same way that subsuming behaviors are triggered by special cases of the triggering stimuli of subsumed behaviors.

The partial world model relevant to a subset of behavioral modules is contained in the module which manages that subset. This is the highest-level module contained in that subset. For example, a behavior called **go-to(elevator)** which goes to the elevator in a building given the current location must direct lower level behaviors to orchestrate a sequence of behaviors (i.e., make managerial decisions) which are triggered at intersections, such as **turn(left)**, **turn(right)**, and **turn(straight)**. The **go-to** behavior contains the necessary world knowledge to get to a known landmark, in this case the elevator. The managed behavior (**turn**) contains no world knowledge of that nature. Its world knowledge is only that which it needs to know to perform a turn successfully.

5. Maze Navigation

Navigation through a maze provides a clear and simple example of the generation and execution of plans. In this case, the successful path through the maze is directly representable as a path through state space, the relevant state in this case being location. A navigation plan is a predetermined sequence describing the choices made at each decision point, i.e., at forks in the road or intersections. This can be represented as a traversal of a decision tree (Fig. 2).

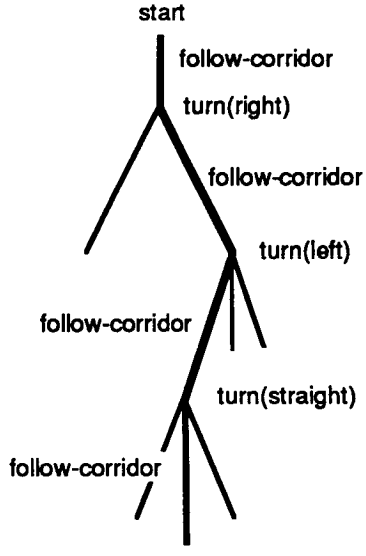


Fig. 2. Maze navigation decision tree

Upon encountering a decision point, such as an intersection, the subsumption architecture relies on a preprogrammed, rote strategy to select one of the alternatives. This arbitrary tie-breaker rule may be "always head south", or "follow the left wall". These can be successful strategies for maze navigation, but they are rarely optimum.

The best plans are based on previous experience. Searches for optimal paths through state space have been studied extensively in artificial intelligence. These require a cost criterion as well as a guide to choices available, i.e., a map. A map simply represents the accumulated experience of the mapmaker. Heuristics may be used (i.e., stay on the main path, or go straight, until you have a good

reason not to) in the absence of specific map knowledge.

This search for an optimum path creates a plan based on the expectations of the planner. If the domain has changed, or the map is incorrect, the planner must generate a new plan. The new plan, in order to be optimal, may require some backtracking, or it may start from the current state[23]. The subsumption planner replans, without explicit backtracking, from the moment it senses a discrepancy between expectation and reality. The plan generated may include as its first segment a backtrack; however, from the point of view of the new plan, it is not backtracking, but rather generating a new complete plan from the current state.

The current location and orientation of the robot is represented as a point in (abstract) state space. Subspaces representing the sensors and their abstractions which are used to determine location, orientation, obstacles, and other properties important to navigation and travel, will contain the triggers for these behaviors. A behavior module is triggered when the current state lies within some specific trigger region. For example, the region of sensor state space which represents the situation in which a large object is directly in front of the robot should trigger an obstacle avoidance behavior module which causes a turn.

5.1 Path planning

A goal is the desired end state of a behavior. This definition of a goal assumes that the robot has achieved the purpose for its existence when this end state has been reached. This is only the case for simple robots and well-defined behaviors. In general, each subtask also has a goal, and each sub-subtask has goals. Similarly, the goal for the entire behavior is really a subgoal of a larger contextual plan, which may only be implicit in the design of a robot. For example, a robot's explicit goal at the topmost level may be to achieve a mowed lawn. This is a subgoal of the implicit task "keep the lawn mowed" which requires that the robot repeat the lawn mowing subtask whenever the grass gets too long. Real plans thus have a hierarchical, self-similar nature where simpler subtasks look very similar to the contextual task. Thus we can treat contextual plans in the same way as subtasks.

The decision tree which represents all possible paths (starting from the current state) to a goal is the problem space within which the navigation plan must be constructed. This problem space is a virtual scratchpad for the subsumption planner. The root of this tree represents the system's current location. The planner visualizes a future course of action which is expected to reach a goal.

The planner makes its decisions based on general knowledge of the problem domain, which it has accumulated by experience or by design. The benefit of using a planner over simple reactive behavior is that knowledge which is more global in nature than that available at the decision point may be applied to solve the problem in (hopefully) a more efficient manner. This more-global knowledge is in fact a partial world model. The premise of the subsumption planner is that partial world models may be distributed across the system as cause-effect predictors, associated with the action-generating behaviors which are capable of steering the robot into a desired state.

It is the responsibility of the cause-effect predictors to propagate the decision tree into the future to determine the best plan. Unlike a game theoretic min-max planner, in which the goal is to win the game, not to reach any specific game state, the subsumption planner can create a much more specific visualization of the goal. A chess playing algorithm can only "visualize" the winning goal state by forward chaining from the current state; there are a great number of possible winning states. It does not select a goal state and attempt to generate a plan which achieves it. The chess player tries a great number of state trajectories until a completely defined state is achieved which belongs to the subset of checkmate (goal) states. To the general purpose planner, however, most specifics of the goal state are irrelevant and can safely be ignored.

In the case of the maze navigator, the only aspect of the goal state which *is* relevant is the location. All other aspects of the goal state can be ignored. Since the goal state may be so loosely defined, the planner can restrict its search for potential behaviors to only those whose effect is to cause the robot to change location.

The most general behavior which can cause this effect may be called the **travel** behavior. This behavior may direct, manage, or simply allow (but not micro-manage!) the action of subservient behaviors which cause forward motion, turns, obstacle avoidance, etc.

The entire course of action of the **travel** behavior need not be planned in great detail in advance. The robot need only realize that **travel** is capable of producing the desired effect. Once this is known, the robot depends on **travel** to complete the entire task of getting to the goal. The repertoire of behav-

iors **travel** has at its beck and call will perform their own planning as needed.

The **travel** behavior makes its decisions in the context of its view of the sensor space. It delegates responsibility for selecting a sequence of turns at intersections to the **navigate-maze** behavior when it senses the domain (walls and passageways) through which it must travel. (An alternate behavior might be **navigate-meadow**; a meadow is a term for a large area where there is a possibility that proximity sensor bearings may be lost, so other navigation techniques, such as dead-reckoning, are called for.)

The **navigate-maze** behavior selects an active subordinate behavior based on its "map" (a representation of its expectations) or on heuristics if the map does not apply to the current situation. It does this by examining relevant elements of the real or virtual state space (orientation and location) and associating with that trigger state a "best" reaction. The "map" is therefore in the form of an associator which generates some reaction based on a set of inputs. This associator represents a partial world model. Each managerial behavior module contains an associator which embodies only the knowledge relevant to that behavior.

The plan, represented as a trajectory in state space, has associated with it a quality indicator. When the state arrives at a decision point in real execution, the decision it makes is the one associated with the highest quality indicator. If sensors indicate a deviation from the expected sensor state as contained in the virtual sensor space, the quality of trajectories which are affected by that deviation is reduced. Alternate plans may or may not be generated at that point. At the point of departure from expectations (i.e., the current state), the robot makes a decision according to the new highest quality plan.

5.2 An Example

Assume that a maze is to be navigated as in Fig. 3. The robot will start at location *s* and is to find its way to goal location *g*. There are two paths. The shortest path, right at *a* and left at *b*, is blocked by an obstacle which cannot be sensed until past *b*.

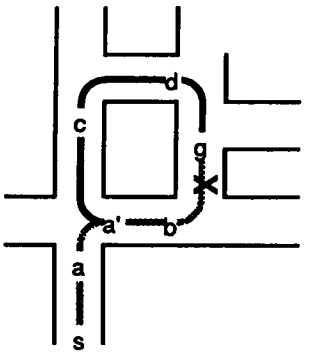


Fig. 3. Maze example

The subsumption planner contains a **travel** behavior capable of physically relocating the robot. This manager behavior controls subordinate behaviors **navigate-maze** and **navigate-meadow**. In the subsumption architecture, it would do this by inhibiting output from the undesired behavior. Another behavior, **follow-corridor**, is a behavior triggered upon sensing walls on either side and clear space in front. It simply travels along the midline of the corridor. The **turn** behavior is a composite behavior, consisting of a **turn-manager** and four types of subordinate turn generating behaviors, **turn(left)**, **turn(right)**, **turn(straight)**, and **turn(back)**. These turns are dependent on orientation; turns dependent on compass direction could be used identically. The **turn-manager** contains a turn-associator which is the mechanism containing the world model for the navigation domain. Since navigation in the maze world is performed by deciding which way to turn at intersections, the turn-map in this case resides in **turn-manager**. Another partial model, the corridor-length-map, provides expectations of distance to the **follow-corridor** predictor. Other low level modules, not discussed here, prevent collisions with obstacles, keep the robot in the center of the hall, keep the robot moving, etc.

The robot will behave as follows:

1) The robot begins at location *s*. The **follow-corridor** reactor begins to generate actions as in the traditional subsumption architecture, and the robot starts moving down the hall. It moves toward location *a*.

2) At the same time that the reactor begins generating actions, the **travel** behavior is triggered by some higher level behavior to start the planning process. The **travel** behavior assumes responsibility for moving the robot from *s* to *g*. The predictor generates a simple state space trajectory between *s* and *g*. **Travel** allows **navigate-maze** to assume re-

sponsibility for completing the plan and its execution. Only **navigate-maze** is triggered by the abstract maze/meadow sensor. The **follow-corridor** sensor is also triggered by the abstract corridor sensor. The **follow-corridor** predictor generates the expected location *a* based on its knowledge of corridor length. In the virtual sensor map, *a* is stored as a waypoint to be reached at time t_q The virtual sensor space at *a* indicates the sensory situation of a 4-way intersection. The **turn-manager** behavior is virtually triggered by an abstract intersection sensor.

3) The **turn-manager** associator senses global location *a*. The global location sensor is abstracted from a variety of fused raw sensors, perhaps including visual clues, wheel odometers, spatial sensors such as sonar or rangefinder, and any other sensors which distinguish this location as different from others in any way. The associator outputs "turn-right" as the highest quality plan, followed by "turn-straight" as the next highest. Note these are not behaviors but rather decisions on which behavior should be activated as part of the plan. Other alternatives have a quality indication of zero.

4) The planner is now done until the robot reaches *a*, since all details for the plan segment from *s* to *a* are complete. The turn manager inhibits its own triggering until ready to resume planning.

5) The robot reaches *a*. Since the prediction turned out to be accurate, the top quality plan has not changed. This plan indicates the behavior **turn(right)** should be executed next. Turn right is selected by **turn-manager** as active, and the predictor generates *a'* as the next waypoint. *a'* virtually triggers **follow-corridor**, and that predictor generates waypoint *b*. Virtual sensor situation *b* then triggers **turn-manager**, which inhibits itself from making a commitment until *b* is physically reached. The planner waits for the robot to catch up.

6) At *b*, the **turn-manager** associator generates the decision "turn-left". Robot starts the **turn(left)** behavior using the reactor mechanism. But, an unexpected obstacle is sensed. The blocked path causes the entire trajectory quality to go to zero. The expectation violation alert is made available to any behavior.

7) The robot is still near *b*. The **navigate-maze** behavior notices the expectation violation. Its reaction is to replan. It predicts that it can still reach the goal, since there is a second, albeit lower quality, plan available from a previous waypoint. All motion-generating behaviors are allowed to trigger from the real sensor state *b*. **Follow-corridor** has the best quality plan, a direct trajectory to end up near *a*. The robot now moves under direction of **follow-corridor**.

8) At *a'*, *c*, and *d*, decisions are made in the same manner as above. Finally, the robot moves to *g*. **Travel** has completed its visualized plan, along with all its subordinates. The **travel** behavior terminates, and current state *g* may be used to trigger the next behavior.

6. Conclusion

This paper has described an extension to the subsumption architecture which implements planning using a virtual sensor space as a planning tool. The planner, given a goal, generates a new plan, monitors execution of the plan, and replans in the event of a discrepancy between expectation and reality. The planner operates as a distributed network in parallel with the existing subsumption network. The theoretical motivation for this technique was presented. An example in the domain of maze navigation showed its operation in a simple application.

References

[1] James S. Albus, *Brains, Behavior, and Robotics*, BYTE Books, Peterborough, NH, 1981.

[2] Marvin Minsky, *The Society of Mind*, Simon and Schuster, Inc., New York, 1986.

[3] Michael A. Arbib, *Schema Theory*, University of Southern California Technical Report 91-03, 1991.

[4] Eugene C. Chalfant and Les Gasser, *Describing Computation and Knowledge using Attractor Superdynamics: Preliminary Report*, University of Southern California Distributed Artificial Intelligence Group Research Note 62, 1990.

[5] Rodney Brooks, *A Robust Layered Control System for a Mobile Robot*, IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1, March 1986.

[6] Rodney Brooks, *Intelligence without representation*, Artificial Intelligence, Vol. 47, pp. 139-159, 1991.

[7] Rodney Brooks, Challenges for Complete Creature Architectures, in *From animals to*

animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior, Jean-Arcady Meyer and Stewart W. Wilson, eds., The MIT Press, Cambridge MA, 1991.

[8] Jonathan H. Connell, *A Behavior-Based Arm Controller*, IEEE Transactions on Robotics and Automation, Vol. 5, No. 6, December 1989.

[9] Jonathan H. Connell, *Minimalist Mobile Robotics: A Colony-style Architecture for an Artificial Creature*, Academic Press, Inc., Boston, 1990.

[10] Maja J. Mataric, *Integration of Representation Into Goal-Driven Behavior-Based Robots*, IEEE Transactions on Robotics and Automation, Vol. 8, No. 3, June 1992.

[11] Marco Dorigo and Uwe Schnepf, *Genetics-Based Machine Learning and Behavior-Based Robotics: A New Synthesis*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 1, January/February 1993.

[12] N. Tinbergen, *The Study of Instincts*, Oxford University Press, Oxford, England, 1966.

[13] D. M. Lyons and A. J. Hendriks, *Planning by Adaptation: Experimental Results*, IEEE International Conference on Robotics and Automation, submitted for review.

[14] Patrick J. Hayes, *The Frame Problem and Related Problems in Artificial Intelligence*, in "Readings in Artificial Intelligence", Bonnie Lynn Webber, Nils J. Nilsson, eds., Tioga Publishing Company, Palo Alto, 1981.

[15] Elaine Rich, *Artificial Intelligence*, McGraw-Hill, New York, 1983.

[16] Edmund H. Durfee and Victor R. Lesser, *Using Partial Global Plans to Coordinate Distributed Problem Solvers*, Proceedings of the 1987 International Joint Conferences on Artificial Intelligence, pp. 875-883, 1987.

[17] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill, *Coherent Cooperation Among Communicating Problem Solvers*, IEEE Transactions on Computers, C-36, pp. 1275-1291, 1987.

[18] Edmund H. Durfee and Victor R. Lesser, *Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No. 5, September/October 1991.

[19] Mitsuo Kawato, Yoshiharu Maeda, Yoji Uno, and Ryoji Suzuki, *Trajectory Formation of Arm Movement by Cascade Neural Network Model Based on Minimum Torque-change Criterion*, Biological Cybernetics, pre-press correspondence, 1990.

[20] Masazumi Katayama and Mitsuo Kawato, *Virtual Trajectory and Stiffness Ellipse During Force-Trajectory Control Using a Parallel-Hierarchical Neural Network Model*, Proceedings of the Fifth International Conference on Advanced Robotics, Italy, 1991.

[21] Makoto Hirayama, Mitsuo Kawato, and Michael I. Jordan, *Speed-Accuracy Trade-off of Arm Movement Predicted by the Cascade Network Model*, ATR Auditory and Visual Perception Research Laboratories Research Note NC89-65, Kyoto, Japan, 1990.

[22] Sukhan Lee, Paul S. Schenker, and Jun Park, *Sensor-Knowledge-Command Fusion Paradigm for Man/Machine Systems*, Proceedings of SPIE International Symposium on Advanced Intelligent Systems, Boston MA, 1990.

[23] Bruce Abramson, *A Decision-Theoretic Framework for Integrating Sensors into AI Plans*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 2, March/April 1993.

AIAA-94-1292-CP

REAL-TIME ROBOT DELIBERATION BY COMPILATION AND MONITORING OF ANYTIME ALGORITHMS

Shlomo Zilberstein
Computer Science Department
University of Massachusetts at Amherst

Abstract

This paper addresses a central issue in robot construction, namely the control of deliberation time. The complexity of automated planning and scheduling makes it undesirable, sometimes infeasible, to find the optimal action in every situation since the deliberation process itself degrades the performance of the system. The question is how can an intelligent robot react to a situation after performing the "right" amount of thinking. It is by now widely accepted that a successful robotic system must trade off between decision quality and the computational resources used to produce it. Anytime algorithms, introduced by Dean, Horvitz and others in the late 1980's, were designed to offer such a trade-off. Recent work by Zilberstein and Russell shows that the advantages of anytime algorithms can be extended to the construction of complex robotic systems. This paper describes the compilation and monitoring mechanisms that are required to build robots that can efficiently control their deliberation time.

I. Control of Deliberation Time

Intelligent robots must perform real-time deliberation to solve such problems as path planning, task scheduling, and interpretation of sensory data. An important aspect of intelligent behavior is the capability of robots to factor the cost of deliberation into the deliberation process. Two factors determine the cost of deliberation: the resources consumed by the process, primarily computation time, and constant change in the environment that may decrease the relevance of the outcome and hence reduce its value. A useful mechanism to quantify this dependency is based on the definition of a utility function $U(S)$ over the states of the world. The utility of a state defines the desirability of that state. For example, the utility of a robot that assembles a certain product can be measured by the number of products completed each hour. Utility functions extend the traditional notion of deadline (allowing for gradual decrease of value over time) and the traditional notion of goals (allowing for partial goal satisfaction). Thus, they are more suitable for control of real-time robotic systems.

To choose an optimal course of action and to maximize its utility function, a robot must perform some real-time problem solving. The performance of robotic systems can be improved by optimizing the quality of their decisions *net* of deliberation cost. The problem of deliberation cost has been widely discussed in economics, engineering and artificial intelligence. In artificial intelligence, researchers have proposed a number of meta-level architectures to control the cost of base-level reasoning [3, 9, 14]. The model presented in this paper belongs to this class of solutions: its meta-level reasoning component optimizes resource allocation to the base-level performance components. This approach separates two, central aspects of robot construction: the development of the performance components and the optimization of performance. This modularity is accomplish by using anytime algorithms as the elementary components of the system.

The rest of the paper describes our approach in detail. Section II describes the notion of anytime algorithms. It shows how to construct anytime algorithms and how to characterize the trade-off that they offer between quality of results and computation time. Section III explains the benefits and difficulties involved in the composition of anytime algorithms. Sections IV and V describe the two main components of our solution to the composition problem, namely off-line compilation and run-time monitoring. Section VI describes briefly some applications of this approach. Finally, Section VII summarizes the benefits of our approach and discusses some directions for further work.

II. Anytime Algorithms

The term "anytime algorithm" was coined by Dean in the late 1980's in the context of his work on time-dependent planning. Anytime algorithms are algorithms whose quality of results improves gradually as computa-

Copyright © American Institute of Aeronautics and Astronautics, Inc., 1994. All rights reserved.

tion time increases, hence they offer a tradeoff between resource consumption and output quality. Many numerical approximation methods, such as Taylor series approximation, are based on iterative improvement and, as such, can be considered an anytime algorithm.

Various metrics can be used to measure the quality of a result produced by an anytime algorithm. From a pragmatic point of view, it may seem useful to define a *single* type of quality measure to be applied to all anytime algorithms. Such a unifying approach may simplify the meta-level control. However, in practice, different types of anytime algorithms tend to approach the exact result in completely different ways. The following metrics have been proved useful in anytime algorithm construction:

1. **Certainty** – this metric reflects the degree of certainty that the result is correct. The degree of certainty can be expressed using probabilities, fuzzy set membership, or any other approach.
2. **Accuracy** – this metric reflects the degree of accuracy or how close is the approximate result to the exact answer. Normally with such algorithms, high quality provides a *guarantee* that the error is below a certain small upper bound.
3. **Specificity** – this metric reflects the level of detail of the result. In this case, the anytime algorithm always produces *correct* results, but the level of *detail* is increased over time.

Many existing programming techniques produce useful anytime algorithms. Examples include iterative deepening search, variable precision logic, and randomized techniques such as Monte Carlo algorithms or fingerprinting algorithms. For a survey of such programming techniques and examples of algorithms see [21].

The notion of interrupted computation is almost as old as computation itself. However, traditionally, interruption was used primarily for two purposes: aborting the execution of an algorithm whose results are no longer necessary, or suspending the execution of an algorithm for a short time because a computation of higher priority must be performed. Anytime algorithms offer a third type of interruption: interruption of the execution of an algorithm whose results are considered "good enough" by their consumer.

Conditional performance profiles

To allow for efficient meta-level control of anytime algorithms, we characterize their behavior by *conditional performance profiles* (CPP) [19]. A conditional performance profile captures the dependency of output quality on time allocation as well as on input quality. In [21], the reader can find a detailed discussion of various types of conditional performance profiles and their representation. To simplify the discussion of compilation, we will refer only to the *expected* CPP that maps computation time and input quality to the expect output quality.

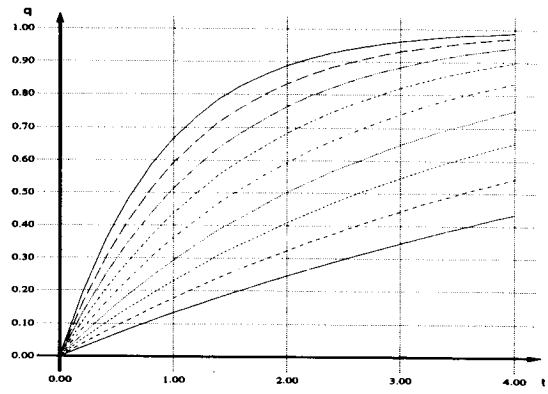


Figure 1: Graphical representation of a CPP

Definition 1 *The conditional performance profile (CPP), of an algorithm A is a function $CPP_A : \mathcal{Q}_{in} \times \mathcal{R}^+ \rightarrow \mathcal{Q}_{out}$ that maps input quality and computation time to the expected quality of the results.*

Figure 1 shows a typical CPP. Each curve represents the expected output quality as a function of time for a *given* input quality.

Interruptible and contract algorithms

In [15] we make an important distinction between two types of anytime algorithms, namely interruptible and contract algorithms. An interruptible algorithm can be interrupted at any time to produce results whose quality is described by its performance profile. A contract algorithm offers a similar trade-off between computation time and quality of results, but it must know the total allocation of time in advance. If interrupted at any point before the termination of the contract time, it may yield no useful results. Interruptible algorithms are in many cases more appropriate for the application, but they are also more complicated to construct. In [15] we show that a simple, general construction can produce an interruptible version for any given contract algorithm, with only a small, constant penalty. This theorem allows us to concentrate on the construction of contract algorithms for complex decision-making tasks and then convert them into interruptible algorithms using a standard transformation.

III. Composing Anytime Algorithms

Modularity is widely recognized as an important issue in system design and implementation. However, the use of anytime algorithms as the components of a modular system presents a special type of scheduling problem. The question is how much time to allocate to each component in order to maximize the output quality of the complete system. We

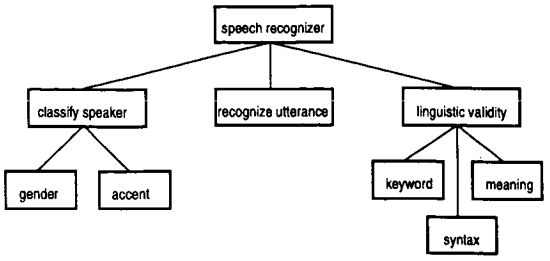


Figure 2: A composite module for speech recognition

refer to this problem as the anytime algorithm *composition problem.*

Consider for example a speech recognition system whose structure is shown in Figure 2. Each box represents an elementary anytime algorithm whose conditional performance profile is given. The system is composed of three main components. First, the speaker is classified in terms of gender and accent. Then a recognition algorithm suggests several possible matching utterances. And finally, the linguistic validity of each possible utterance is determined and the best interpretation is selected. The composition problem is the problem of calculating how much time to allocate to each elementary component of the composite system, so as to maximize the quality of the utterance recognition.

Solving the composition problem is important for several reasons. First, it introduces a new kind of modularity into real-time system development by allowing for separation between the development of the performance components and the optimization of their performance. In traditional design of real-time systems, the performance components must meet certain time constraints that are not always known at design time. The result is a hand-tuning process that, may or may not, culminate with a working system. Anytime computation offers an alternative to this approach. By developing performance components that are responsive to a wide range of time allocations, one avoids the commitment to a particular performance level that might fail the system.

The second reason why the composition problem is important relates to the difficulty of programming with anytime algorithms. To make a composite system optimal (or even executable), one must control the activation and interruption of the components. In solving the composition problem, our goal is to minimize the responsibility of the programmer regarding this optimization problem. Our solution is described in the following two sections.

IV. Compilation

Given a system composed of anytime algorithms, the compilation process is designed to: (a) determine the optimal performance profile of the complete system; and (b) insert into the composite module the necessary code to achieve that performance. The precise definition and solution of the problem depend on the following factors:

1. **Composite program structure** – what type of programming operators are used to compose anytime algorithms?
2. **Type of performance profiles** – what kind of performance profiles are used to characterize elementary anytime algorithms?
3. **Type of anytime algorithms** – what type of elementary anytime algorithms are used as input? what type of anytime algorithm should the resulting system be?
4. **Type of monitoring** – what type of run-time monitoring is used to activate and interrupt the execution of the elementary components?
5. **Quality of intermediate results** – what access does the monitoring component have to intermediate results? is the actual quality of an intermediate result known to the monitor?

Depending on these factors, different types of compilation and monitoring strategies are needed. To simplify the discussion in this paper, we will consider only the problem of producing contract algorithms when the conditional performance profiles of the components are given. We will assume that no active monitoring is allowed once the system is activated. A broader, in-depth analysis of compilation and monitoring can be found in [21].

Let $\mathcal{F}$ be a set of anytime functions. Assume that all function parameters are passed by value and that functions have no side-effects (as in pure functional programming). Let $\mathcal{I}$ be a set of input variables. Then, the notion of a composite expression is defined as follows:

Definition 2 *A **composite expression** over $\mathcal{F}$ with input $\mathcal{I}$ is:*

1. *An expression $f(i_1, ..., i_n)$ where $f \in \mathcal{F}$ is a function of n arguments and $i_1, ..., i_n \in \mathcal{I}$.*
2. *An expression $f(g_1, ..., g_n)$ where $f \in \mathcal{F}$ is a function of n arguments and each g_i is a composite expression or an input variable.*

For example, the expression $A(B(x), C(D(y)))$ is a composite expression over $\{A, B, C, D\}$ with input $\{x, y\}$. Suppose that each function in $\mathcal{F}$ has a conditional performance profile associated with it that specifies the quality of its output as a function of time allocation to that function and the qualities of its inputs. Given a composite expression of size n, the main part of the compilation process is to determine a mapping:

$$\mathcal{T} : t \rightarrow (t_1, ..., t_n) \quad (1)$$

This mapping determines for each total allocation, t, the allocation to the components that maximizes the output quality.

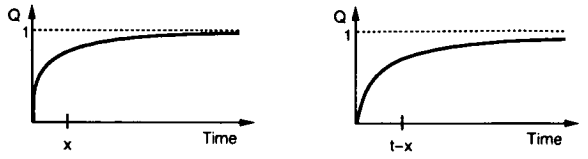


Figure 3: The performance profiles of $\mathcal{A}_1$ and $\mathcal{A}_2$

A compilation example

Let us look first at a simple example of compilation involving only two anytime algorithms. Suppose that one algorithm takes the input and produces an intermediate result. This result is then used as input to another anytime algorithm which, in turn, produces the final result. Many systems can be implemented by a composition of a sequence of two or more algorithms. For example, an automated repair system can be composed of two algorithms: diagnosis and treatment. This can be represented in general by the following expression:

$$Output \leftarrow \mathcal{A}_2(\mathcal{A}_1(Input))$$

Figure 3 shows the performance profiles of $\mathcal{A}_1$ and $\mathcal{A}_2$. These performance profiles are defined by:

$$Q_1(t) = 1 - e^{-\lambda_1 t} \qquad Q_2(t) = 1 - e^{-\lambda_2 t}$$

Assume that the output quality is the sum of the qualities of $\mathcal{A}_1$ and $\mathcal{A}_2$, then the following result holds:

Theorem 3 *Given the performance profiles of $\mathcal{A}_1$ and $\mathcal{A}_2$, the optimal time allocation mapping is:*

$$\mathcal{T} : t \rightarrow \left(\frac{\ln \lambda_1 - \ln \lambda_2 + \lambda_2 t}{\lambda_1 + \lambda_2}, \frac{\ln \lambda_2 - \ln \lambda_1 + \lambda_1 t}{\lambda_1 + \lambda_2}\right) \quad (2)$$

Proof: Since the overall output quality is:

$$Q(x) = 1 - e^{-\lambda_1 x} + 1 - e^{-\lambda_2 (t-x)} \quad (3)$$

the maximal quality is achieved when $\frac{\partial Q}{\partial x} = 0$.
In other words:

$$\lambda_1 e^{-\lambda_1 x} - \lambda_2 e^{-\lambda_2 (t-x)} = 0 \quad (4)$$

The solution of this equation yields the above allocation. □

To complete the compilation process, the compiler needs to insert code in the original expression for proper activation of $\mathcal{A}_1$ and $\mathcal{A}_2$ as contract algorithms with the appropriate time allocation. This is done by replacing the simple function call by an anytime function call [21]. The implementation of an anytime function call depends on the particular programming environment and will not be discussed in this paper.

The complexity of compilation

The compilation problem is defined as an optimization problem, that is, a problem of finding a schedule of a set of components that yields maximal output quality. In order to analyze its complexity, it is more convenient to refer to the decision problem variant of the compilation problem. Given a composite expression e, the conditional performance profiles of its components, and a total allocation B, the decision problem is whether there exists a schedule of the components that yields output quality greater than or equal to K. To begin, consider the general problem of global compilation of composite expressions, or GCCE. In [21], we prove the following result:

Theorem 4 *The GCCE problem is NP-complete in the strong sense.*

The proof is based on a reduction from the PARTIALLY ORDERED KNAPSACK problem which is known to be NP-complete in the strong sense. The meaning of this result is that the application of the compilation technique may be limited to small programs. To address the complexity problem of global compilation, we developed an efficient local compilation technique.

Local compilation

Local compilation is the process of finding the best performance profile of a module based on the performance profiles of its *immediate* components. If those components are not elementary anytime algorithms, then their performance profiles are determined using local compilation. Local compilation replaces the global optimization problem with a set of simpler, local optimization problems and reduce the complexity of the whole problem. Unfortunately, local compilation cannot be applied to every composite expression. If the expression has repeated subexpressions, then computation time should be allocated only once to evaluate all identical copies. Local compilation cannot handle such cases. However, the following three assumptions make local compilation both efficient and optimal [21]:

1. **The tree-structured assumption** – the input composite expression has no repeated subexpressions, thus its DAG (directed acyclic graph) representation is a tree.
2. **The input-monotonicity assumption** – the output quality of each module increases when the quality of the input improves.
3. **The bounded-degree assumption** – the number of inputs to each module is bounded by a constant, b.

Under these assumptions, local compilation is both efficient and yields optimal results [21]. The first assumption is needed so that local compilation can be applied. The second assumption is needed to guarantee the optimality of the resulting performance profile. And the third assumption is needed to guarantee the efficiency of local compilation. Using an efficient tabular representation of performance profiles, we could perform local compilation in constant

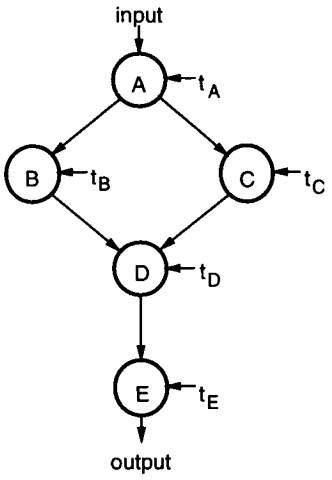


Figure 4: DAG representation of F

time and reduce the overall complexity of compilation to be linear in the size of the program.

Repeated subexpressions

While the input-monotonicity and the bounded-degree assumptions are quite reasonable (and also desirable from a methodological point of view), the tree-structured assumption is somewhat restrictive. We want to be able to handle the case of repeated subexpressions. To understand the problem, consider the following expression:

$$F = E(D(B(A(x)), C(A(x))))$$

Figure 4 shows the DAG representation of F. Recall that the purpose of compilation is to compute a time allocation mapping that would specify for each input quality and total allocation of time the best apportionment of time to the components so as to maximize the expected quality of the output. But local compilation is only possible when one can repeatedly break a program into sub-programs whose execution intervals are disjoint, so that allocating a certain amount of time to one sub-program does not affect in any way the evaluation and quality of the other sub-programs. This property does not hold for DAGs. In the example shown in Figure 4, B and C are the ancestors of D, but their time allocations cannot be considered independently since they both use the same sub-expression, $A(x)$.

To address this problem we have developed a number of *approximate* compilation techniques that work efficiently on DAGs, but do not guarantee optimality of the schedule [21]. The compilation of additional programming constructs, such as conditional statements and loops, is analyzed in [21]. To summarize, a number of compilation techniques have been developed that can efficiently produce the performance profile of a composite system based on the performance profiles of its components.

V. Run-Time Monitoring

Monitoring plays a central role in anytime computation as it complements anytime algorithms with a mechanism that determines their run-time. We examine the monitoring problem in two types of domains. One type is characterized by the predictability of utility change over time. High predictability of utility allows an efficient use of contract algorithms modified by various strategies for contract adjustment. The second type of domains is characterized by rapid change and a high level of uncertainty. In such domains, active monitoring, that schedules interruptible algorithms based on the value of computation criterion, becomes essential.

Given a compound anytime program, $\mathcal{P}$, whose elementary anytime components are $E = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$, a monitoring scheme is defined as a mapping that determines a certain time allocation for each activation of an elementary component.

Definition 5 *A **monitoring scheme** for a program $\mathcal{P}$ is a mapping:*

$$\mathcal{M} : E \times Z^+ \rightarrow R^+$$

where E is the set of elementary components of $\mathcal{P}$.

$\mathcal{M}(i,j)$ is the time allocation to the j^{th} activation of the i^{th} component. A monitoring scheme supplies the necessary information to make a compound anytime program executable in a well defined way. In defining monitoring schemes, we make a distinction between *passive* and *active* monitoring.

Definition 6 *A monitoring scheme is said to be **passive** if the corresponding time allocation mapping is completely determined prior to the activation of the system.*

Definition 7 *A monitoring scheme is said to be **active** if it is not passive. That is, the corresponding time allocation mapping is partially determined while the system is active.*

Under active monitoring, some scheduling decisions are made at run-time. Such decisions are based on the *actual* quality of results produced by the anytime components and based on the *actual* change that occurred in the environment. The main reason why active monitoring is necessary in control of anytime algorithms is the problem of uncertainty. In an entirely deterministic world, passive monitoring can yield optimal performance. However, in unpredictable domains there is much to be gained in performance by introducing an active monitoring component.

Two primary sources of uncertainty affect the operation of real-time robotic systems. The first source is internal to the system. It is caused by the unpredictable behavior of the system itself. The second source is external. It is caused by unpredictable changes in the environment. These two sources of uncertainty are characterized by two separate

knowledge sources. Uncertainty regarding the performance of the system is characterized by the performance profile of the system (in particular, we use *performance distribution profiles* to represent the probability distribution of quality of results). Uncertainty regarding the future state of the environment is characterized by the model of the environment. Obviously, the type of active monitoring may vary as a function of the source of uncertainty and the degree of uncertainty.

Monitoring contract algorithms

It is easier to construct contract algorithms than interruptible ones, both as elementary and as compound algorithms. Therefore, I will examine first the monitoring problem assuming that the complete system is presented as a contract algorithm, $\mathcal{A}$. The conditional performance profile of the system is $Q_{\mathcal{A}}(q,t)$ where q is the input quality and t is the time allocation. Assume that $Q_{\mathcal{A}}(q,t)$ represents, in the general case, a probability distribution. When a discrete representation is used, $Q_{\mathcal{A}}(q,t)[q_i]$ denotes the probability of output quality q_i.

Let S_0 be the current state of the domain and let S_t represent the state of the domain at time t, let q_t represent the quality of the result of the contract anytime algorithm at time t. $U_{\mathcal{A}}(S,t,q)$ represents the utility of a result of quality q in state S at time t. This utility function is given as part of the problem description. The purpose of the monitor is to maximize the expected utility of the result, that is, to find t for which $U_{\mathcal{A}}(S_t,t,q_t)$ is maximal. Contract algorithms are especially useful in a particular type of domains which is defined as follows:

Definition 8 *A domain is said to have* ***predictable utility*** *if $U_{\mathcal{A}}(S_t,t,q)$ can be determined for any future time, t, and quality of results, q, once the current state of the domain, S_0, is known.*

The notion of predictable utility is a property of domains. The same utility function can be predictable in one domain and unpredictable in another. What makes a domain predictable is the capability to determine the exact value of results of a particular quality at any future time. Hence, the state of the domain may change, even in an unpredictable way, and utility may still be predictable. To explain this situation, we define a function, $f(S)$, that isolates the features of a state that determine its utility. In other words,

$$\forall S_1,S_2 \;\; f(S_1)=f(S_2) \;\Rightarrow\; U_{\mathcal{A}}(S_1,t,q)=U_{\mathcal{A}}(S_2,t,q) \tag{5}$$

Consider for example a transportation domain that refers to traffic on a particular road. The state of the domain is defined by the location and velocity of each vehicle and $f(S)$ may be, for example, the traffic density. Using the function f, it is easy to show that a domain with predictable utility is a domain for which $f(S_t)$ can be determined once the current state, S_0, is known. In general, three typical cases of such domains can be identified:

1. A static domain is obviously predictable since $S_t = S_0$ and $f(S_t) = f(S_0)$. For example, the game of chess constitutes a static domain.
2. A domain that has a deterministic model is predictable since future states can be uniquely determined and hence $f(S_t)$ can be determined. For example, a domain that includes moving objects has a deterministic model when the velocity of each object is constant.
3. A domain for which there is a deterministic model to compute $f(S_t)$, once the current state is known, is predictable. Note that this does not require a deterministic model of the domain itself. An important sub-class is all the domains for which $f(S)=\emptyset$, that is, domains in which the utility function depends only on time.

The initial contract time

The first step in monitoring contract algorithms involves the calculation of the initial contract time. Due to uncertainty concerning the quality of the result of the algorithm, the expected utility of the result at time t is represented by:

$$U'_{\mathcal{A}}(S_t,t)=\sum_i Q_{\mathcal{A}}(q,t)[q_i]U_{\mathcal{A}}(S_t,t,q_i) \tag{6}$$

The probability distribution of future output quality is provided by the performance profile of the algorithm. Hence, an initial contract time, t_c, can be determined before the system is activated by solving the following equation:

$$t_c = arg\max_t\{U'_{\mathcal{A}}(S_t,t)\} \tag{7}$$

Under passive monitoring, this initial contract time is used to determine (using the compiled performance profile of the system) the ultimate allocation to each component.

In some cases, it is possible to separate the value of the results from the time used to generate them. In such cases, one can express the comprehensive utility function, $U_{\mathcal{A}}(S,t,q)$ as the difference between two functions:

$$U_{\mathcal{A}}(S_t,t,q)=V_{\mathcal{A}}(S_0,q)-Cost(S_0,t) \tag{8}$$

where $V_{\mathcal{A}}(S,q)$ is the value of a result of quality q in a particular state S (termed *intrinsic utility* [14]) and $Cost(S,t)$ is the cost of t time units provided that the current state is S. Similar to the expected utility, the expected intrinsic utility for any allocation of time can be calculated using the performance profile of the algorithm:

$$V'_{\mathcal{A}}(S,t)=\sum_i Q_{\mathcal{A}}(q,t)[q_i]V_{\mathcal{A}}(S,q_i) \tag{9}$$

Finally, the initial contract time can be determined by solving the following equation:

$$t_c = arg\max_t\{V'_{\mathcal{A}}(S_0,t)-Cost(S_0,t)\} \tag{10}$$

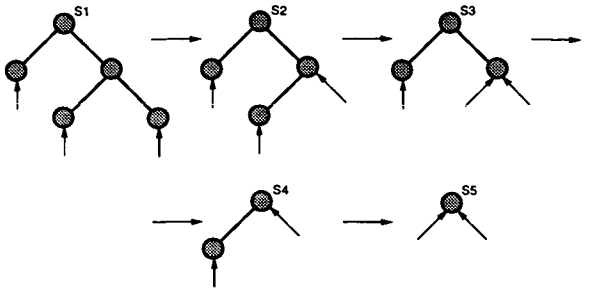


Figure 5: A sequence of residual sub-systems

Once an initial contract time is determined, several monitoring policies can be applied. The most trivial one is the *fixed-contract* strategy that leads to a passive monitoring scheme. Under this strategy, the initial contract time and the compiled performance profile of the system are used to determine the allocation to the components. This allocation remains constant until the termination of the problem solving episode. The fixed-contract policy is optimal under the following conditions:

Theorem 9 ***Optimality of monitoring of contract algorithms.*** *The fixed-contract monitoring strategy is optimal when the domain has predictable utility and the system has a fixed performance profile.*

Proof: This result is rather trivial since, when the domain has predictable utility and the system's performance profile is fixed, utility of results at any future time can be determined. The initial contract time, that maximizes the comprehensive utility, remains the same during the computation and no additional scheduling decision can improve the performance of the system. □.

We now look at two extensions to the fixed-contract policy for cases with high degree of uncertainty regarding the quality of the results. In such cases, the initial contract time must be altered by an active monitoring component.

Re-allocating residual time

The first type of active monitoring that we analyze involves reallocation of residual time among the remaining anytime algorithms. Suppose that a system, composed of several elementary contract algorithms, is compiled into an optimal compound contract algorithm. Since the results of the elementary contract algorithms are not available during their execution, the only point of time where active monitoring can take place is *between* activations of the elementary components. Based on the structure of the system, an execution order can be defined for the elementary components. The execution of any elementary component can be viewed as a transformation of a node in the graph representing the program from a computational node to an external "input" of a certain quality. This transformation is shown in Figure 5. The quality of the new input is only known when the corresponding elementary component terminates. Based on the actual quality, the remaining time (with respect to the global contract) can be reallocated among the remaining computational components to yield a performance improvement with respect to allocation that was based on the probabilistic knowledge of quality of intermediate results.

In order to be able to allocate time optimally to each component, the monitor needs to access not only the performance profile of the complete system, but also the performance profiles of the residual sub-systems. The compilation problem has to be solved for each residual system. For example, for the system modeled by Figure 5, five performance profiles must be calculated. These performance profiles can be derived using the standard local compilation technique. The only difference is that the compiler does not need to store the allocation to all the components but only the allocation to the next component in the activation order.

Adjusting contract time

The second type of active monitoring for contract algorithms involves adjustments to the original contract time. As before, once an elementary component terminates, the monitor can consider its output as an input to a smaller residual system composed of the remaining anytime algorithms. By solving the previous equation that determines the contract time for the residual system, a better contract time can be determined that takes into account the actual quality of the intermediate results generated so far.

If the elementary components are interruptible, the contract time can be adjusted *while* an elementary component is running. Given the quality of the results generated by that component and its performance profile, a new contract may be determined. In that case, the new contract may affect the termination time of the currently active module in addition to affecting the run-time of future modules.

Monitoring interruptible algorithms

We turn now to the problem of monitoring interruptible anytime computation. The use of interruptible algorithms is necessary in domains whose utility function is not predictable (and cannot be approximated by a predictable utility function). Such domains are characterized by nondeterministic rapid change. Medical diagnosis in an intensive care unit, trading in the stock exchange market, and vehicle control on a highway are examples of such domains. Many possible events can change the state of such domains and the timing of their occurrence is essentially unpredictable. Consequently, accurate projection into the far future is very limited and the previous fixed-contract approach fails. Such domains require interruptible decision making.

Active monitoring using the value of computation

Consider a system whose main decision component is an interruptible anytime algorithm, $\mathcal{A}$. The conditional probabilistic performance profile of the algorithm is $Q_{\mathcal{A}}(q,t)$ where q is the input quality and t is the time allocation. As before, $Q_{\mathcal{A}}(q,t)$ is a probability distribution and $Q_{\mathcal{A}}(q,t)[q_i]$ denotes the probability of output quality q_i.

Let S be the current state of the domain. Let S_t be the state of the domain at time t. And, let q_t represent the quality of the result of the interruptible anytime algorithm at time t. $U_{\mathcal{A}}(S,t,q)$ represents the utility of a result of quality q in state S at time t. The purpose of the monitor is to maximize the expected utility by interrupting the main decision procedure at the "right" time. Due to the high level of uncertainty in rapidly changing domains, the monitor must constantly assess the value of continued computation by calculating the net expected gain from continued computation given the current best results and the current state of the domain. This is done in the following way:

Due to the uncertainty concerning the quality of the result of the algorithm, the expected utility of the result in a given future state S_t at some future time t is represented by:

$$U'_{\mathcal{A}}(S_t,t) = \sum_i Q_{\mathcal{A}}(q,t)[q_i]U_{\mathcal{A}}(S_t,t,q_i) \qquad (11)$$

The probability distribution of future output quality is provided by the performance profile of the algorithm. Due to the uncertainty concerning the future state of the domain, the expected utility of the results at some future time t is represented by:

$$U''_{\mathcal{A}}(t) = \sum_S p(S_t = S)U'_{\mathcal{A}}(S,t) \qquad (12)$$

The probability distribution of the future state of the domain is provided by the model of the environment.

Finally, the condition for continuing the computation at time t for an additional Δt time units is therefore $VOC > 0$ where:

$$VOC = U''_{\mathcal{A}}(t+\Delta t) - U''_{\mathcal{A}}(t) \qquad (13)$$

Similar to the case of contract algorithms, monitoring of interruptible systems can be simplified when it is possible to separate the value of the results from the time used to generate them. In such cases, one can express the comprehensive utility function, $U_{\mathcal{A}}(S,t,q)$, as the difference between two functions:

$$U_{\mathcal{A}}(S_t,t,q) = V_{\mathcal{A}}(S,q) - Cost([t_c,t]) \qquad (14)$$

where $V_{\mathcal{A}}(S,q)$ is the intrinsic utility function, S is the current state, t_c is the current time, and $Cost([t_c,t])$ is the cost of the time interval $[t_c,t]$. Under this separability assumption, the intrinsic value of allocating a certain amount of time t to the interruptible system (resulting in domain state S) is:

$$V'_{\mathcal{A}}(S,t) = \sum_i Q_{\mathcal{A}}(q,t)[q_i]V_{\mathcal{A}}(S,q_i) \qquad (15)$$

Hence, the intrinsic value of allocating a certain time t in the current state is:

$$V''_{\mathcal{A}}(t) = \sum_S p(S_t = S)V'_{\mathcal{A}}(S,t) \qquad (16)$$

And the condition for continuing the computation at time t for an additional Δt time units is again $VOC > 0$ where:

$$VOC = V''_{\mathcal{A}}(t+\Delta t) - V''_{\mathcal{A}}(t) - Cost([t,t+\Delta t]) \qquad (17)$$

Theorem 10 ***Optimality of monitoring of interruptible algorithms.*** *Monitoring interruptible algorithms using the value of computation criterion is optimal when $\Delta t \to 0$ and when the intrinsic value function is monotonically increasing and concave down and the time cost function is monotonically increasing and concave up.*

Proof: A function q is called *concave up* on a given interval I if it is continuous, piecewise differentiable, and $\forall x,y \in I$ for which $q'(x)$ and $q'(y)$ exist, $(x < y) \Rightarrow (q'(x) \leq q'(y))$. It is called *concave down* if $\forall x,y \in I$ for which $q'(x)$ and $q'(y)$ exist, $(x < y) \Rightarrow (q'(x) \geq q'(y))$. Note that the assumption of monotonically increasing and concave down intrinsic value function is identical to the assumption of Dean and Wellman (See [4], Chapter 8, page 364) that performance profiles have the property of *diminishing returns.*

Now, suppose that the current time is t_1 and that

$$VOC = V''_{\mathcal{A}}(t_1+\Delta t) - V''_{\mathcal{A}}(t_1) - Cost([t_1,t_1+\Delta t]) \leq 0 \qquad (18)$$

Since the intrinsic value function is concave down, it is guaranteed that for any future time $t_2 > t_1$:

$$V''_{\mathcal{A}}(t_2+\Delta t) - V''_{\mathcal{A}}(t_2) \leq V''_{\mathcal{A}}(t_1+\Delta t) - V''_{\mathcal{A}}(t_1) \qquad (19)$$

Since the time cost function is concave up, it is guaranteed that for any future time $t_2 > t_1$:

$$Cost([t_2,t_2+\Delta t]) \geq Cost([t_1,t_1+\Delta t]) \qquad (20)$$

Hence, it is guaranteed that for any future time t_2:

$$VOC = V''_{\mathcal{A}}(t_2+\Delta t) - V''_{\mathcal{A}}(t_2) - Cost([t_2,t_2+\Delta t]) \leq 0 \qquad (21)$$

And therefore termination at the current time is an optimal decision. □

Summary

The monitoring problem has been examined in two types of domains. One type is characterized by the predictability of utility change over time. High predictability of utility allows an efficient use of contract algorithms modified by various strategies for contract adjustment. The

second type of domain is characterized by rapid change and a high level of uncertainty. In such domains, monitoring must be based on the use of interruptible algorithms and the value of computation criterion. In domains with moderate change and some degree of predictability of future utility, one can use an integrated approach. That is, activate the system with an initial contract time and then, if additional time is available, continue to monitor it using the value of computation criterion.

VI. Applications

The advantages of compilation and monitoring of anytime algorithms have been demonstrated through a number of applications. In this section we briefly describe two such application.

Mobile robot navigation

One of the fundamental problems facing any autonomous mobile robot is the capability to plan its own motion using noisy sensory data. A simulated robot navigation system has been developed by composing two anytime modules [22]. The first module, a vision algorithm, creates a local domain description whose quality reflects the probability of correctly identifying each basic position as being free space or an obstacle. The second module, a hierarchical planning algorithm, creates a path between the current position and the goal position. The quality of a plan reflects the ratio between the shortest path and the path that the robot generates when guided by the plan.

Anytime hierarchical planning is based on performing coarse-to-fine search that allows the algorithm to find quickly a low quality plan and then repeatedly refine it by replanning a segment of the plan in more detail. Hierarchical planning is complemented by an execution architecture that allows for the execution of abstract plans – regardless of their arbitrary level of detail. This is made possible by using plans as advice that direct the base level execution mechanism but does not impel a particular behavior. In practice, uncertainty makes it impossible to use plans except as a guidance mechanism.

The conditional performance profile of the hierarchical planner is shown in Figure 6. Each curve shows the expected plan quality as a function of run-time for a particular quality of the vision module. Finally, an active monitoring scheme was developed to use the compiled performance profile of this system and the time-dependent utility function of the robot in order to allocate time to vision and planning so as to maximize overall utility.

One interesting observation of this experiment was that the anytime abstract planning algorithm produced high quality results (approx. 10% longer than the optimal path) with time allocation that was much shorter (approx. 30%) than the total run-time of a standard search algorithm. This

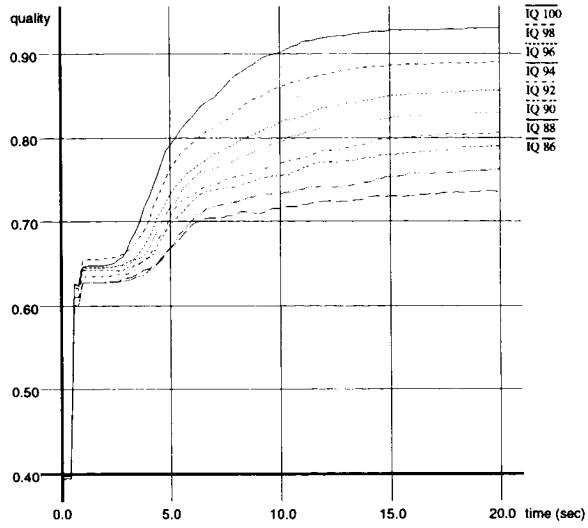


Figure 6: The CPP of the anytime planner

shows that the flexibility of anytime algorithms does not necessarily require a compromise in overall performance.

Model-based diagnosis

Model-based diagnostic methods identify defective components in a system by a series of tests and probes. Advice on informative probes and tests is given using diagnostic hypotheses that are based on observations and a model of the system. The goal of model-based diagnosis is to locate the defective components using a small number of probes and tests.

The General Diagnostic Engine [5] (GDE) is a basic method for model-based diagnostic reasoning. In GDE, observations and a model of a system are used in order to derive *conflicts* (A conflict is a set of components of which at least one has to be defective). These conflicts are transformed to *diagnoses* (A diagnosis is a set of defective components that might explain the deviating behavior of the system). The process of observing, conflict generation, transformation to diagnoses, and probe advice is repeated until the defective components are identified. GDE has a high computational complexity – $O(2^n)$, where n is the number of components. As a result, its applicability is limited to small-scale applications. To overcome this difficulty, Bakker and Bourseau have developed a model-based diagnostic method, called Pragmatic Diagnostic Engine (PDE), whose computational complexity is $O(n^2)$. PDE is similar to GDE, except for omitting the stage of generating all diagnoses before determining the best measurement-point. Probe advice is given on the basis of the most relevant conflicts, called *obvious* and *semi-obvious* conflicts (An obvious (semi-obvious) conflict is a conflict that is computed using no more than one (two) observed outputs).

In order to construct a real-time diagnostic system, Pos [13] has applied the model of compilation of anytime algorithms to the PDE architecture. PDE can be analyzed as a composition of two anytime modules. In the first module, a subset of all conflicts is determined. Pos implements this module by a contract form of breadth-first search. The second module consists of a repeated loop that determines which measurement should be taken next, takes that measurement and assimilates the new information into the current set of conflicts. Finally, the resulting diagnoses are reported.

Two versions of the diagnostic system have been implemented: one by constructing a contract algorithm and the other by making the contract system interruptible using our reduction technique. The actual slow down factor of the interruptible system was approximately 2, much better than the worst case theoretical ratio of 4.

VII. Conclusion

We presented a model for intelligent robot control that is based on compilation and monitoring of anytime algorithms. It offers both a methodological and a practical contribution to the field of real-time deliberation. The main aspects of this contribution include: (1) simplifying the design and implementation of complex intelligent robots by separating the design of the performance components from the optimization of performance; (2) mechanizing the composition process and the monitoring process; and (3) constructing machine independent real-time robotic systems that can automatically adjust resource allocation to yield optimal performance.

The study of anytime computation is a promising and growing field in artificial intelligence and in real-time systems. Some of the primary research directions in this field include: (1) Extending the scope of compilation by studying additional programming structures and producing a large library of anytime algorithms; (2) Extending the scope of anytime computation to include the two other aspects of robotic systems, namely sensing and action; and (3) Developing additional, larger applications that demonstrate the benefits of this approach. The ultimate goal of this research is to construct robust real-time systems in which perception, deliberation and action are governed by a collection of anytime algorithms.

References

[1] M. Boddy and T. L. Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 979–984, Detroit, Michigan, 1989.

[2] M. Boddy. Anytime problem solving using dynamic programming. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 738–743, Anaheim, California, 1991.

[3] T. L. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 49–54, Minneapolis, Minnesota, 1988.

[4] T. L. Dean and M. P. Wellman. *Planning and Control.* San Mateo, California: Morgan Kaufmann, 1991.

[5] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence* **32**:97–130, 1987.

[6] A. Garvey and V. Lesser. Design-to-time real-time scheduling. To appear in *IEEE Transactions on Systems, Man and Cybernetics*, 1993.

[7] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, Seattle, Washington, 1987.

[8] E. J. Horvitz, H. J. Suermondt and G. F. Cooper. Bounded conditioning: Flexible inference for decision under scarce resources. In *Proceedings of the 1989 Workshop on Uncertainty in Artificial Intelligence*, pp. 182–193, Windsor, Ontario, 1989.

[9] E. J. Horvitz and J. S. Breese. *Ideal partition of resources for metareasoning.* Technical Report KSL-90-26, Stanford Knowledge Systems Laboratory, Stanford, California, 1990.

[10] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* **27**: 97–109, 1985.

[11] V. Lesser, J. Pavlin and E. Durfee. Approximate processing in real-time problem-solving. *AI Magazine* **9**(1):49–61, Spring 1988.

[12] R. S. Michalski and P. H. Winston. Variable precision logic. *Artificial Intelligence* **29**(2):121-146, 1986.

[13] A. Pos. *Time-Constrained Model-Based Diagnosis.* Master Thesis, Department of Computer Science, University of Twente, The Netherlands, 1993.

[14] S. J. Russell and E. H. Wefald. Principles of metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, R.J. Brachman *et al.* (eds.), San Mateo, California: Morgan Kaufmann, 1989.

[15] S. J. Russell and S. Zilberstein. Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 212–217, Sydney, Australia, 1991.

[16] W-K. Shih, J. W. S. Liu and J-Y. Chung. Algorithms for scheduling imprecise computations with timing constraints. *SIAM Journal on Computing*, **20**(3):537–552, 1991.

[17] H. A. Simon. *Models of bounded rationality, Volume 2.* Cambridge, Massachusetts: MIT Press, 1982.

[18] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton, New Jersey: Princeton University Press, 1947.

[19] S. Zilberstein and S. J. Russell. Efficient resource-bounded reasoning in AT-RALPH. In *Proceedings of the First International Conference on AI Planning Systems*, pp. 260–266, College Park, Maryland, 1992.

[20] S. Zilberstein and S. J. Russell. Constructing utility-driven real-time systems using anytime algorithms. In *Proceedings of the IEEE Workshop on Imprecise and Approximate Computation*, pp. 6–10, Phoenix, Arizona, 1992.

[21] S. Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms.* Ph.D. dissertation, Department of Computer Science, University of California at Berkeley, 1993.

[22] S. Zilberstein and S. J. Russell. Anytime sensing, planning and action: A practical model for robot control. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1402–1407, Chambery, France, 1993.

AIAA-94-1293-CP

Passive Mapping and Intermittent Exploration for Mobile Robots

Sean P. Engelson
Yale University
Department of Computer Science
P.O. Box 208285 Yale Station
New Haven, CT 06520-8285
Email: engelson@cs.yale.edu

Abstract

An autonomous robot must be able to learn maps of its environment while accomplishing meaningful tasks. Such 'passive' map-learning is difficult, since it cannot rely on active exploration. However, it gives the robot more flexibility in how it learns about a complex novel environment. The primary difficulty is that incorrect maps may be inferred, since insufficient information may be available at any one time (since exploration is disallowed). We address this problem by allowing maps to contain errors, while correcting those errors when possible via a set of heuristic error-correction strategies. Results in a realistic simulation with a random walk (to simulate worst-case explorative action) demonstrates the efficacy of the basic technique.

Passive mapping may still be inefficient, so we incorporate intermittent exploration while maintaining the benefits of the passive approach. This is done by using predefined 'opportunity scripts' which direct the robot in ways that improve the map. Scripts are short plans which are applied depending on whether or not they interfere with other robot goals. The scripts we have implemented mostly use known techniques to improve mapper efficiency. Occasional use of these scripts over a base random walk strategy shows a speedup in map convergence, demonstrating the efficacy of intermittent exploration. Also, by using intermittent exploration, very complex worlds could be learned efficiently.

1 Motivation

While robotic manipulation technology has revolutionized manufacturing in recent years, the potentials of mobile robotics remain virtually unused. There are a number of reasons for this, some of them sociological, but the main reason is that the technology of mobile robotics is not yet sufficiently mature for most practical applications. One of the main research areas which requires development is *map-learning*. By this, we mean the automatic acquisition by a mobile robot of a model of its large scale environment (floor layout, for example). This is needed, even in known environments, because hard-wiring the structure of its environment into a robot can be very costly, and it is difficult (if not impossible) to keep such a representation up to date when the environment is changed. Naturally, there are also cases where the structure of the environment cannot be known accurate in advance even to the system designers; automated mapping is required in those cases.

There are many useful tasks that mobile robots can perform that require the use of some sort of environmental map. One example is that of an office or factory courier, whose function is to transfer materials between areas of a building. This task is one of the few for which mobile robots are currently being used; TRC has a robotic courier installed in hospitals, for delivering non-critical items to patients upon request. Their robot contains a detailed, preprogrammed map of the hospital building it works in; the hospital environment was also engineered somewhat to support reliable navigation (beacons were places, for example). Effective map-learning would cut the costs of installing such a system, by both easing the initial setup, and by making the robots more flexible (they could be transferred between different buildings, say).

Another class of tasks are those in which little or no a priori information about the environment exists. One such task, ripe for robot use, is search-and-rescue missions. Rescuing people often involves going into hazardous environments, so it would be desirable to use robots wherever possible. Such rescue missions, whether from burning buildings or from caves, often require learning the structure of the environment on the fly, so that searching is done efficiently (time is usually of the essence). Also, there is no time to explore for exploration's sake (we will return to this point below). A task which is similar, though in a completely different domain, is planetary exploration. Getting men to Mars to carry out scientific exploration and experimentation is, at present, a difficult proposition. A cost-effective solution that has been proposed is to send robots to gather scientific data. These robots will need to move about in large areas of the surface to gather that data, and hence will need some sort of internal mapping to support navigation.

1.1 Passive mapping

There are several features of the tasks described above that underscore important issues for robotic map learning. The first is that mapping does not occur in a vacuum. It is a part of a larger system, aiding navigational planning. A related point is that mapping should take place during normal goal-directed task execution. In fact, a 'mapping phase' wherein the agent maps out its entire environment may be wholly infeasible due to the environment's size or changeability. We therefore propose the view that mapping should be 'passive', and not require controlling the robot's actions. It thus functions as a static map, as far as planning is concerned, but the map's utility transparently

Copyright©1993 by the American Institute of Aeronautics and Astronautics, Inc.
All rights reserved.

improves over time. A schematic diagram of the high-level cognitive architecture our view implies is shown in Figure 1. The mapper and deliberator (action selector) are functionally independent. Exploration enters deliberation through a sort of 'back door', via suggested exploration scripts (described below in Section 5).

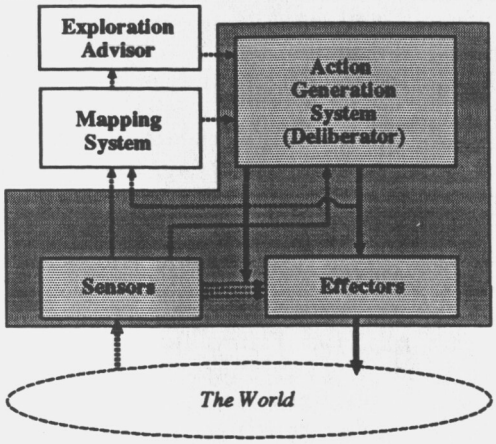


Figure 1: Cognitive modular divisions in the passive mapping paradigm. Solid arrows indicate control flow; dashed arrows indicate information flow.

Our approach may be contrasted with much previous research in map learning, where the mapper is viewed as a procedure which, after some amount of time, outputs a 'correct' map; this representation is then to be used for planning and reasoning. This paradigm, or variations thereof, is ubiquitous (eg, [3; 4]). These methods typically require the mapper to be 'active' and to take control of the robot when uncertain information must be verified. This is needed due to the desire to learn a 'complete' and 'correct' map in finite time; however, this active approach interferes with goal-achievement. This approach has other problems in the real world as well, since the real world is (practically) open-ended—the world to be learned cannot be bounded. Also, attention must be paid, in constructing a world representation, to the use to which it will be put. We address these issues below.

1.2 Overview

In the next section, we describe a general framework for developing adaptive models of a robot's environment, which we have used to develop our mapping system. We then describe the representation scheme we use for mapping, which incorporates both topological and metric information. In Section 4, we describe the algorithms used in the mapping system, including mapping-error diagnosis and correction methods. Given this passive mapper, we then develop methods for intermittent exploration, to improve mapping efficiency. We then describe our results in a robotic simulation. Section 7 reviews related previous work in map-learning. We close with a discussion of how our methods could be used in practice, and future directions for our research.

2 Adaptive Modeling

2.1 Adaptive State-Space Models

We view the mapping system as a sort of *adaptive state-space model*. There are two fundamental operations for which a state-space model is used: state estimation and state prediction. Planning uses these operations in conjunction with a domain theory describing the physics of the world. We may thus view such a model as a black box which outputs a state prediction given a previous state and a robot action, and outputs an improved state estimate given a state prediction and sensory input. This is essentially what is known to control theorists as an *observer*. It gives a very general conceptual framework, encompassing both continuous estimation methods such as the Kalman filter and discrete methods such as Markov chain models. The fundamental point we wish to stress is the use of the model as a black box for estimation and prediction, as far as the rest of the planning/control system is concerned. Internal issues of representation, and indeed whether or how the representation changes over time, should be largely irrelevant to the rest of the system. In what follows, we develop an architecture for such systems, and then show how we have applied it to the specific problem of mobile robot map-learning.

2.2 Discretizing the World

Robots typically live in continuous state-spaces (eg, the plane for a mobile robot); to represent these spaces effectively, some sort of discretization must be done. Our work focuses on robots designed to operate in indoors environments (office buildings, factories, etc.). We adopt a technique based on Kuipers' topological mappers [13; 14], which use control laws with good stability properties (actions) to pick out distinguished 'places' in a continuous space. This allows a distinguished set of points (actually, small regions) to constitute 'waypoints'. Waypoints may be corridor intersections, or areas near particular pieces of equipment; the main requirement is that they be recognizable. Waypoints can be used to represent the structure of the entire space, relative to the capabilities of the robot. As noted by Kuipers, this approach allows a representation to directly support navigational planning. Abstractly, the state space is represented as a finite state machine with non-deterministic transitions (which can often be assigned transition probabilities).

2.3 The Architecture

We now present an architecture for adaptive discrete state-space models. We first divide the system at a high level into an *estimator* and an *adapter* (refer to Figure 2). The core of the estimator is the loop between projection and matching. The projector predicts new states given old state estimates and control inputs. There may be multiple state estimates in the system, which we call *tracks* (each tracks a possible true state). The matcher decides which states are consistent with each predicted estimate, given the current perceptual input. Thus far, we have the standard recursive estimation framework. Now, the spaces we are interested in are both very large (thousands of states) and relatively unstructured (without even ap-

proximate closed form estimators). Hence we need indexing, to find likely candidate states to feed to the matcher. Further, in different situations, different matching methods will be appropriate (for error recovery, for example). Therefore, the matcher is divided into a general matching engine, dependent only on the representation language, and a task-dependent set of matching methods (*matchers*). The matching engine also provides a method of conflict resolution to determine which matchers are applicable in given circumstances.

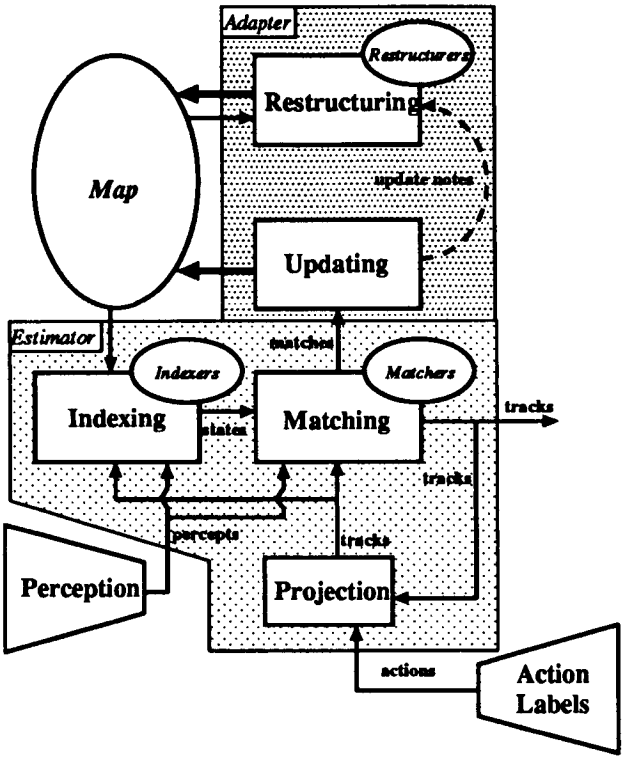


Figure 2: Adaptive state-space model architecture.

The adapter consists of an updating module and a restructuring module. The updater adjusts the parameters of states in the representation, for example, the position of a known waypoint. It also can monotonically change the topology of the represented state space by, for example, adding new state transitions. The exact type of updating that is performed depends on the matches found by the matcher; different matchers may (and usually will) have different associated update methods. A deeper sort of adaptation is restructuring, which uses information about the large-scale structure of the known state-space as well as integration of observations over longer terms to adjust the structure of the state-space. This may involve splitting or coalescing states, adjusting hierarchical relationships, and so on. Like matching, and for the same reason, we divide this module into a general restructuring engine and a task-dependent set of restructurers. To alleviate the problem of searching aimlessly in the state space for restructuring to do, it can be advantageous for the updater to inform the restructuring module when it performs an update, since a change made by the updater to the representation may trigger a restructuring operation.

3 Diktiometric Representation

The first step in specializing the architecture described above to robot mapping is the specification of a representation language for our maps. The straightforward method using the discrete state-space approach amounts to topological mapping, the method suggested by Kuipers in [13]. The world is represented as a graph of waypoint nodes, labeled with local perceptual information. Transitions are labeled with robot control routines which constitute the actions that move the robot between waypoints ([9] describes how these routines are derived). We extend this notion to directly take into account geometric knowledge as well. *Diktiometric*[1] representation explicitly represents geometric relations between waypoints. A diktiometric representation (a *diktiometry*) consists of two graphs, a *path graph* and a *reference graph* (see Figure 3). Path graph nodes represent waypoints, and arcs represent action transitions. Nodes in the reference graph represent local coordinate frames, with links giving known geometric relations. The two graphs are connected by *reference links*, giving the positions of waypoints with respect to particular local reference frames.

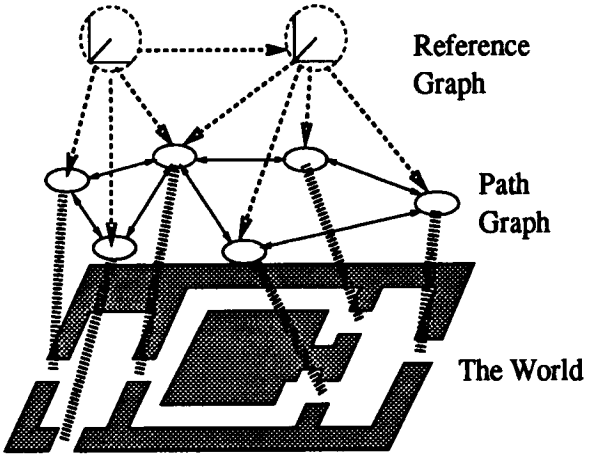


Figure 3: A schematic picture of a diktiometry, where doorways are taken as 'waypoints'. Action links are solid, reference links are dashed.

3.1 Uncertain Geometry

We represent uncertain geometric relations between waypoints relative to local reference frames with limited coverage. This ensures that, provided the robot knows which reference frame's domain it is in, relative uncertainly remains bounded. This is so regardless of the method used to represent uncertainty. In the multi-dimensional Gaussian distribution approach [20], the local reference frame

[1] From the Greek *δίκτυον* meaning 'network', and *μέτρον* meaning 'measurement'. Diktiometric representations represent the world as a network of waypoints with relative positions.

approach amounts to maintaining a set of smaller covariance matrices (with smallish eigenvalues) instead of a single large covariance matrix for the entire system (which will necessarily have some large eigenvalues, hence large uncertainty, even between nearby points). The local reference frames are related to each other by small covariance matrices; a position in any frame can be related to any other by appropriate composition. The local method is also a performance win when locality can be established (though doing so may incur extra costs). Rather than using such a statistical representation for odometric uncertainty, however, we advocate the use of bounding intervals.

The primary reason that Gaussian noise models are so widely used is the Kalman filter, and its non-linear cousin, the extended Kalman filter. These are the optimal linear recursive updating procedures, given a truly Gaussian noise model. However, when this model is not correct, the Kalman filter can be suboptimal and can even diverge. We therefore prefer a non-distributional approach. Rather than representing a probability distribution on the true value of a measurement, we give bounds on the possible true values. These bounds give us a set of possible values within which true must lie. Projection and updating can be done easily and efficiently using interval arithmetic [1]. Our contention is that odometry is subject to so many unmodellable sources of noise (slippage, bumps, voltage fluctuations, etc.) that the best that can be hoped for is reasonable bounds on relative position. In a typical office environment, with current equipment, odometric error over distances of several meters is about 10% of distance traveled[2]. This gives reasonably tight bounds on relative motion; use of sensory feedback (eg, visual motion analysis) may also help.

4 The Mapping System

The adaptive state-space architecture described above, combined with diktiometric representation, provides a framework for designing a robot mapping system which supports the flexible navigation planning tasks we wish to address. This section describes in more detail how this works.

4.1 Indexing waypoints

For some applications using the adaptive state-space model paradigm, indexing is trivial, due to there being a relatively small number of states. If, however, we wish to learn diktiometries of large-scale spaces in an open-ended fashion, we must deal with thousands of waypoints (at least). It is simply infeasible to examine the entire diktiometry for matching waypoints to extend a track. On the other hand, there is no fool-proof way of generating only the most 'similar' candidates that we know of. Hence, we developed heuristic indexing methods that should work well in practice. There are three categories of indexing we examine: *expectational*, *geometric*, and *perceptual*.

Each of the geometric and perceptual indexing modules produces a stream of sets of candidate waypoints. Each set in a stream contains better candidates than those following it, while members of a set are assumed to all be equivalently good. The indexing methods are integrated by running the modules in parallel and combining the candidate sets that come out.

Expectations The simplest form of indexing uses the path graph to predict the expected state of the robot after executing the current action. Given a particular current waypoint, the expectational candidates are those which are predicted by the last action taken from that waypoint. Naturally, there may be more than one, since actions can be non-deterministic. Expectational indexing produces one candidate set, used together with the first sets produced by other indexing.

Geometric Indexing The second sort of indexing is geometric indexing, based on waypoint positions. The basic idea is to find waypoints whose position relative to a track's is consistent with the last position change. We will discuss two indexing methods which use the reference graph to find waypoints likely to be near the current (projected) robot position.

The simplest method is to use a depth-limited search through the reference graph, starting from the current track's frame. If the new position estimate may be consistent with one of a frame's waypoints, the waypoints are checked individually for consistency and candidates suggested. This method assumes that the area has been reasonably well explored, so that the robot moves between frame regions known to be neighbors. This implies that each ply of the search is less plausible (as the search gets farther from the source), giving, as desired, a stream of candidate sets. On the other hand, the assumption of nearby frames giving correct candidates will not always be correct, particularly in the early stages of learning (however, more frames may be searched then, since there is less to search). On the other hand, the farther a frame is in the reference graph, the less useful information (in general) it gives for constraining the robot's position.

Perceptual Indexing The objective of perceptual indexing is to quickly find those stored percepts that are most similar to the current one. Furthermore, since the robot will never be in quite the same configuration twice, the indexing method should be robust with respect to small changes in position and orientation. We have developed an image-based method for waypoint recognition, using the notion of *image signatures*. An image signature is an array of values, each computed by a *measurement function* from a subset of the image (the image is tesselated). As demonstrated in [8], signatures can be matched to each other for fairly reliable recognition. The problem here is how to index this database so that similar signatures taken at different orientations will be found quickly. This can be done by indexing the signatures by their columns, since if two signatures are taken at different rotations, they will match at a horizontal offset. Hence, an input signature's columns are used to index the columns close to them, marking each signature found at the offset implied by the column match. When enough of a database signature's columns have been marked, the signature's waypoint is suggested as a match candidate.

Column indexing can easily be implemented as a k-d tree [18]; an iteratively growing hypercube search is used

[2] Jonathan Connell, personal communication.

to search progressively further and further from the input column. In this way, good candidate waypoints can be found based on perceptual cues, even if they are geometrically distant. A similar approach could also be taken using 3D estimates of the positions of visual features. Using the method of Atiya and Hager [2], feature triples can be represented by a set of 6 parameters describing a triangle. If triples of features going around the robot's viewpoint are stored thusly, a similar k-d tree approach can be used for indexing. Using this method, more sophisticated methods of 3D recognition and position registration can be done as well, given the required perceptual capabilities (eg, a system such as [21]).

4.2 Matching and updating

State identification in diktiometric mapping amounts to matching the robot's projected position and sensory inputs with candidate waypoints (generated as above). If we could guarantee that no mapping errors would ever occur, then the matching problem amounts to little more than filtering possibilities for consistency. However, as noted above, this is never the case, and so error diagnosis and correction must continually be performed. One way in which this is done is through the use of multiple matchers, each indicating a particular state of affairs, including mapping errors. Each matcher consists of a match test which compares the projected robot state with waypoints in a candidate set, and an update method which is applied to waypoints that are determined to match. Resolving between multiple applicable matchers is done by arranging them in a partial order; the maximal applicable matchers are used. This preference relation is constructed so that more reasonable decisions take precedence over less likely decisions (posit as few and as plausible errors as possible). If no matchers apply, a new waypoint node is created.

4.2.1 Matchers

There are four main matchers that we currently use. Two that correspond to normal extension of the map are CONTINUE and LINK. CONTINUE matches a waypoint that was expected with consistent position estimate and view; the waypoint's position estimate and view set are updated. LINK matches a waypoint with consistent position estimate and view which was unexpected, and so also adds a new action link to the path graph. Two other matchers correct for positional inconsistency caused by incorrect waypoint identification or odometric error. To deal with the case where a waypoint's position interval does not contain its true position (due to update with an outlier), the system keeps track of the waypoint's *nominal envelope*, the least bounding interval of the estimates used for updating the waypoint's position. This is asymptotically guaranteed to contain the true position. Thus, E-MATCH matches a waypoint such that the projected robot position is consistent with the waypoint's nominal envelope, indicating a possible waypoint position inconsistency. The waypoint's position estimate is the grown to contain the nominal envelope, presumably consistent. To deal with a track drifting from true, N-MATCH allows a match with a waypoint near the track's projected position; the track is then 'snapped' to the waypoint. A fifth, 'pseudo-match' is used for waypoint creation; when it is chosen to be used, a new waypoint is added with a track's projected state.

4.2.2 Dynamic priorities

A static priority scheme for matchers, while easy to implement, will seldom really be appropriate. For example, CONTINUE should only be preferred to LINK in well-explored areas, otherwise it is no better (since few links are known). We thus propose a dynamic scheme for prioritizing matchers, using estimates of the quality of the mapper's knowledge. This is done using local grid-based methods to maintain estimates of how well areas have been visited or traversed, as well as confidence in each individual track, based on its recent history. We can thus express preferences as the following:

- Prefer CONTINUE to LINK if the region just traversed has been well traversed (hence probably mapped) in the past.
- Prefer E-MATCH or waypoint creation to N-MATCH when a track has high confidence, and the converse when a track has low confidence.

In a similar fashion, other ideas of when particular types of matching are more appropriate can be expressed. This framework of a dynamic partial order controlled by meta-knowledge determined preferences seems both flexible enough to encode the required diagnostic knowledge, while providing a simple and modular mode of expression.

4.3 Transients

The first function of the restructurer in our system is to deal with map errors due to *transients*, non-existent states and transitions hallucinated due to nonreproducable conditions. Of course, if a hallucination is consistent, it may (usually) be considered real enough, as far as the robot is concerned. The only way to detect these transients, without taking over control of the robot, is to maintain statistics of when each link is thought to have been traversed and each waypoint is thought to have been visited. If these are compared with the frequency that the link or waypoint was expected, transients will be distinguished by a very low frequency of occurrence. The offending link/waypoint is then removed from the diktiometry. This also helps deal with (slowly) changing environments.

4.4 Waypoint restructuring

A deeper sort of error that cannot be discovered by using imprecise matching (hence requiring restructuring) is *structural inconsistency*. Incorrect waypoint identification can lead to either multiple waypoints in the world being represented by a single waypoint node (*polytopy*) or the converse, a single real-world waypoint represented by multiple nodes (*monotopy*). These sorts of errors can be dealt with by the system's restructurers. The concept behind these restructurers is that while one observation of a waypoint may not be sufficient to determine its identity, integration of many observations can yield statistically significant evidence of environmental structure. We deal below with two restructurers, one for *splitting* to deal with polytopy, and one for *merging* to deal with monotopy. To a great extent these two are inverses, and we apply similar methods for both, as summarized in Table 1. We divide the constraints applied into three categories:

Constraints	Splitting	Merging
Geometric	Multimodal distribution of position observations for a waypoint node	Unimodal distribution of position observations for two different waypoint nodes
Local	Separable correlated sets of input-output action link pairs	Nearly identical output link sets
Non-local	—	Multiple mergable track histories

Table 1: Constraints for waypoint restructuring.

geometric constraints on the positions of waypoints, *local path* constraints about local functional relationships in the path graph, and *non-local path* constraints applied to larger portions of the path graph.

Each of these restructurers can, in principle, operate independently. To integrate them, we propose to use a 'veto-based' strategy. Each restructuring method depends on certain thresholds, giving the desired confidence in a diagnosis of mono/polytopy. If we give each two thresholds, such that when the higher is satisfied we say that a diagnosis is *proposed*, and when the lower is not satisfied the diagnosis is *rejected*, the three strategies can be used in tandem as follows. If a diagnosis of either monotopy or polytopy is proposed by at least one restructurer, and is not rejected by any, we accept the diagnosis. This provides a sensible and modular integration of multiple constraints for diktiometry restructuring.

Geometric constraints The essential insight here is that by making the reasonable assumption that all waypoints are at least some minimum distance apart (call it δ_{sep}), we can discover when sets of position observations (from odometry) come from one or many waypoints. If then make the further weak statistical assumption that the distribution of position estimates for a waypoint is unimodal, we can use a multimodality test to test for structural inconsistency. If a single waypoint node represents two real waypoints, we would expect the distribution of position estimates matched to the node to be bimodal. Similarly, if two nodes correspond to one waypoint, their combined observation set should be unimodal. This will not always work, so we use other constraints as well.

Local path constraints The next sort of restructuring we consider uses a functional kind of constraint. The idea is that each waypoint has a consistent, if stochastic, input-output behavior (effects of performing actions). This being the case, polytopy is indicated by inconsistent effects at one waypoint, and monotopy by two waypoints with (nearly) identical effects. This is similar to Chrisman's approach to the closely related *perceptual aliasing problem* found in reinforcement learning [6]. Here, the merging method is simpler—if two waypoints have nearly identical incoming and outgoing action link sets (greater than a large fraction go to the same waypoint via equivalent actions), then the waypoints should be merged. The waypoints also should have been visited often enough to ensure that the known links are representative. Splitting requires examining how well incoming links predict outgoing links. If the incoming links can be partitioned into two sets such that the sets' 'images' (the sets of outgoing links taken after coming in on each link in a set) are (nearly) disjoint, then a split is indicated. This heuristic detects cases where a waypoint node does not adequately represent a functional state of the robot. By assumption, each waypoint corresponds to a single functional state (though the converse need not hold).

Non-local path constraints A third method uses non-local path constraints for determining restructuring. Currently, this only applies to merging. One problem with the local path approach to diagnosing monotopy is *merging deadlock*, where two different waypoints both need to be merged, but each inhibits the other being merged since local information does not suffice (the link sets are not identical). Hence what is needed is a sort of sub-graph isomorphism. Unfortunately, this is intractable, so we use a heuristic approximation. As the robot travels through the world, each track maintains a history of the waypoints it matched to. As well, each waypoint P in a history is associated with other waypoints that (a) were considered as matches but rejected, and (b) could be mergable with P (ie, their position estimates are consistent). An arbitrary length limit is placed upon histories to prevent them from growing without bound. When a track matches a waypoint, it deposits copies of its histories at the waypoint. When two waypoints have enough histories consistent with each other, the histories are used to 'sew up' a portion of the path graph. This may introduce spurious merges, but with conservative threshold choices, it can work in concert with the two methods described above to provide effective restructuring.

5 Opportunistic Exploration

Even though passive mapping, as described above, is important so that the robot can pursue its goals while learning, it can also be inefficient. This problem can be ameliorated somewhat, without sacrificing the benefits of passivity, by allowing the mapper to *advise* the deliberator of actions that the mapper would find useful. These can be treated by the deliberator as goals to satisfy when feasible; failure of a mapper goal does not invalidate a plan. So, if the robot decides it has an extra ten minutes before it has to deliver a package, it can take a short trip down a side corridor to see where it leads. The main point is that ultimate control lies inside the deliberator, which has the responsibility of balancing the utility of the various goals it must achieve.

We implement this idea by using *opportunity scripts*—short, stereotyped sequences of actions designed to help mapping in particular situations (an early version of this is described in [11]). These are suggested to the deliberator by an *opportunity checker*, which examines the current mapper state to determine if any scripts are applicable. Those which are, are sent to the deliberator where they may get executed. Even if they aren't, there is no great loss, since the mapping system's correctness does not depend on the actions the robot takes. There are two

If:	• Uncertainty of the last Δposition high • Uncertainty of all track positions high • Didn't just retrace step
Do:	• Try to get a fixation on the last waypoint visited; • If found, go there, otherwise exit; • Try to return, if possible.

RETRACE STEP

If:	• There is a nearby waypoint whose uncertainty is high • Only one track is current
Do:	• Try to get a fixation on the uncertain waypoint based on its relative position; • If fixation found, go there, otherwise exit.

HEAD FOR UNCERTAINTY

If:	• There is only one track, with high uncertainty • There is nearby waypoint with low uncertainty
Do:	• Try to go to that waypoint.

HEAD FOR CERTAINTY

sorts of opportunities that can be dealt with in this way—exploration and experimentation.

There are two reasons to use opportunity scripts to aid mapping. One is that a robot will not naturally explore the world efficiently, since its actions are determined by other considerations. The other is that mapper-directed activity may improve the reliability of mapping decisions and reduce the introduction of errors into the map. We have developed and tested some heuristic opportunity scripts to thus aid mapping; they are described below.

5.1 Exploration scripts

The essential idea of using opportunity scripts to improve mapping is that they can be performed whenever a high-level decision process determines that other goals can be put off for a short while. This implies, first of all, that these scripts must apply in general circumstances, as the mapper has no control over when they will be called upon. Secondly, the scripts must be fairly limited in duration, so that they can do their business of improving mapping and then let the robot get back to its high-level goals. Scripts have two components, an application test which determines if the script is relevant, and a (loop-less) procedure which is executed if the script is applied. Scripts use the mapper's data structures, in particular the set of current tracks and the map itself.

We have investigated a number of exploration scripts. Some seek to reduce positional uncertainty. Others attempt to find new waypoints and action links. Scripts are also used to reduce ambiguity in the map or in the robot's position estimate. Some probe map waypoints which seem likely to not really exist. Keep in mind that all these scripts do is direct the behavior of the robot, indirectly focusing the attention of mapping system.

RETRACE STEP:

One important source of error and ambiguity in a map is positional uncertainty. When the robot performs an action with a very uncertain estimate of relative position, and the robot's *a posteriori* position estimate (after matching to its map) is also particularly uncertain, a simple and useful heuristic for reducing uncertainty both in the map and in the current position estimate is to retrace the last step taken. That is, the robot tries to return to the waypoint it just came from; if it manages that, it tries to get back to where it started.

HEAD FOR UNCERTAINTY:

A more generally applicable heuristic for reducing map uncertainty is to simply head for nearby waypoints with large positional uncertainty, under the assumption that if they are reached, new constraints will improve the positional estimate. This can only be reasonably tried, of course, if the robot's current waypoint is unambiguous.

HEAD FOR CERTAINTY:

The converse of the last script is useful when the robot's positional uncertainty gets too high, and that is to head for a nearby waypoint whose position is known very precisely. If the robot reaches and recognizes the waypoint, the robot's position will then also be known more precisely, improving further mapping.

DISAMBIGUATE TRACKS:

It will often occur that the robot's estimate of its current position will be ambiguous. If there are thus multiple current tracks, a good way to distinguish between them is to try to perform an action with different results for the different possible waypoints the robot is at. This will usually result in the incorrect track becoming inconsistent and thus dropped.

PROBE AMBIGUOUS ACTION:

One kind of map ambiguity is when a waypoint has multiple action links coming from it labeled with the same action. While this ambiguity may be inherent, it may also be that one of the links is due to a transient; hence, further examination is warranted. This is achieved by attempting to perform such an ambiguous action—this will tend to speed up elision of any transients, and just maintain the real action links.

PROBE SPLITTAGE:

The kind of map ambiguity we consider for now is where two identical links from different waypoints end at the same waypoint and there is reason to believe that only one is real. This happens when a waypoint is split (see

If:	• There are multiple current tracks.
Do:	• Choose an action link whose destination is known reachable from some, but not all, of the tracks' waypoints; • Try to go to that link's destination waypoint.

DISAMBIGUATE TRACKS

If:	• There is a single current track, • There are multiple links from the current waypoint labeled with the same action.
Do:	• Perform that action.
PROBE AMBIGUOUS ACTION	

If:	• There is a single current track, • The current waypoint was recently split off from another waypoint.
Do:	• Choose an action link which both waypoints have in common, • Try to traverse it.
PROBE SPLITTAGE	

above). Since both waypoints resulting from a split have copies of the same action links, many of those links will be invalid. Hence, when the robot is at a waypoint which resulted from a recent split, the PROBE SPLITTAGE script tries to perform an action that has (in the map) identical consequences in the two split-off waypoints. This will accelerate elision of the invalid action links.

HEAD FOR UNEXPLORED AREA:

Most of the previous scripts have dealt with improving the system's knowledge of waypoints already in its map. Finding new, unknown waypoints in an efficient manner, however, would also be useful, so that the world may be more quickly explored. We thus try to head for an area of the world which has not likely been visited by the robot before. To decide that this holds of some area, each local reference frame has associated with it a *coverage grid*, which tesselates the area about the frame into a coarse grid, and keeps track of an estimated certainty that the robot has visited each grid cell. Then, an area is deemed to be unexplored if the likelihood of part of it having been visited in the past is sufficiently low. Thus, HEAD FOR UNEXPLORED AREA looks in the vicinity of the current waypoint for a nearby area which looks unexplored, and if one is found, attempts to head in its direction.

HEAD FOR RARE WAYPOINTS:

Recall that transient environmental features are eventually elided by the mapping system by noting their frequency of apprehension. This process can be speeded up if the robot tries to reach waypoints which look likely to be transients. Since transients, by their very nature, are not encountered often, it is likely that a waypoint that has not been visited often is transient. If so, then repeatedly trying to reach the waypoint will cause the system to notice that it is not encountered as expected, and so it will eventually be elided. If it is not a transient, then the mapper will just gain a bit more information about the waypoint.

5.2 Experimentation

The main type of experimentation that can be done in our framework delays diktiometry adaptation until more information has come in. Whenever a radical update or restructuring would normally be performed, a note is made of the operation to be performed, along with the information required before it can actually be performed and a set of exploration scripts to help gather that information. For example, before performing a merge, a script may be used to probe the distinctness of the two waypoints. This information is associated with the waypoint(s) involved, so that when the robot returns, the scripts are then suggested to the deliberator for execution. Again, as with exploration scripts, the particulars of the experimentation scripts used depend on the types of updating and restructuring done. We are currently working on developing a set of experimentation scripts for our system, but they have not yet been implemented.

If:	• There is only one current track. • The current waypoint has a neighbor which has been visited less than a threshold number of times.
Do:	• Choose such a rarely visited neighbor, • Try to go to there.
HEAD FOR RARE WAYPOINTS	

6 Results

We present here the results of some experiments we have performed in simulation on the system described above. The simulator is described in [10]; space precludes a full discussion here. Briefly, waypoints are determined by configurations of walls and image signatures are simulated by noisy samples of wall 'color'. Wherever possible, worst-case assumptions were made with respect to sensor and effector noise; all such parameters are adjustable. The performance of the mapper was quantitatively evaluated by measuring *a posteriori* position error—the error inherent in allowing the robot to rely on the map to determine its expected position after each move. We measure this by calculating the sum-of-squared-distance (SSD) between the robots actual relative motion and predicted relative motion for each track after a move. If the system is effective at mapping, we expect the average SSD per move to asymptotically converge to a small constant. There are other useful performance metrics discussed in [9], but the different methods give qualitatively similar results—space does not permit inclusion here.

Figure 4(a) shows a typical small environment used for evaluation. The world was designed to be confusing; every waypoint looks the same as every other. For each run, the robot was controlled by an essentially random walk, while the mapper ran in the background. A move is defined as a sequence of actions ending with the robot in a distinguished waypoint (here, corners or doorways). Each run was 700 moves; a good maps were generally learned within 300. As Figure 4(b) shows, SSD position error starts out high, but quickly begins to converge to a low asymptote (non-zero due to inherent odometric error). This demonstrates the effectiveness of the system in a confusing environment, even with no mapper control over the robot.

The use of exploration scripts were tested by randomly executing them when applicable (generally 30% of the time). Comparative results are shown in Figure 4(c), which shows a significant improvement in mapper performance

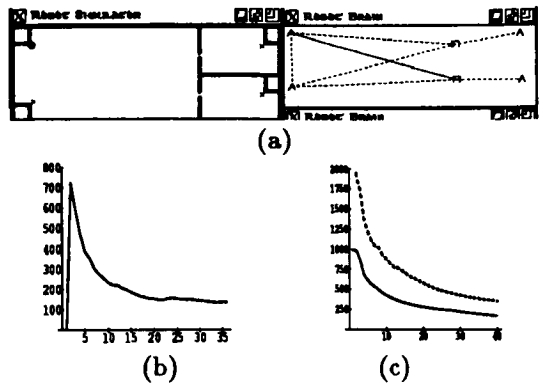


Figure 4: (a) A simple, but confusing, robot environment (left) and a typical map learned (right). (b) SSD position error (y-axis) vs. number of waypoint visits (x-axis), averaged over 10 runs. (c) Improvement (in another, larger, world) by occasionally using exploration scripts (solid) vs. pure random walk (dashed).

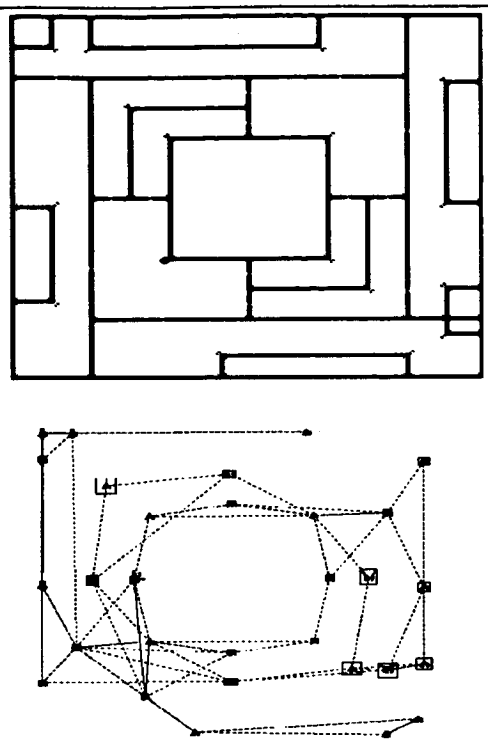


Figure 5: A complicated world and a learned map, learned after 3000 moves, with exploration scripts.

when exploration scripts are used. We expect a further improvement when we have taken into account the conflict resolution between different scripts; we are currently investigating how to compare scripts so that the most useful gets executed. This question is connected to the larger question of assigning utility to exploration and experimentation, which is important in terms of the deliberator's tradeoffs between goal-achievement and mapper script execution. This will form an important focus of our work in the near future.

7 Related Work

Much research on navigational mapping deals with problems of local metric representation (eg., [3; 7; 2]), which is generally unsuitable in large-scale spaces. Kuipers and Byun first develop the notion of a topological place graph based on 'distinctive' locations [14]. However, while they go to some length to avoid error creeping into the map by using active experimentation, there is no provision for error correction. Levitt *et al.* also utilize a topological map which avoids accumulation of navigation error, by using local reference frames based on landmarks [15]. However, their system appears to depend heavily on reliable landmark acquisition. Miller and Slack use information generated for reactive local navigation to build rough geometrical maps of rocky terrain [17]. Their maps are notable in that they can directly be used for reactive navigation.

Basye *et al.* develop a probabilistic theoretical framework for map learning [4]. They probabilistically eliminate errors in the learned map by using active exploration, assuming limited directional certainty and globally recognizable places. However, their methods use very simple models of perception and action and do not use the rich geometrical and perceptual structure available. Hence they are forced to use strongly active strategies to learn maps reliably.

Our methods for dealing with mapping errors can also be incorporated into existing mapping systems with minimal modification. Virtually any system that uses a place graph representation can be reformulated in our terms. Specifically, Sarachik's system for visual navigation [19] is particularly apposite. Her system visually recognizes room shapes and finds doors, linking them together in a place graph. A room's perceived shape can be used as its perceptual description; its position can be described in a locally determined reference frame. Our error correction machinery could then be applied virtually as is.

Mataric [16] uses constraints derived from knowledge of the robot's underlying behavior to derive a topological map based on linear graph segments. Yeap [22] describes a hierarchical topological map, with place nodes described by 2D geometric models. Braunegg [5] develops a similar style of map, where rooms are characterised by the geometric arrangement of vertical edges, measured by stereo vision. Kriegman [12] describes a method for visually instantiating generic models of the robot's surroundings, such as hallways, bringing top-down constraints to bear on geometric interpretation.

8 Discussion

In this paper, we have shown how map-learning can be organized around the principle of passive mapping. Passive mapping involves two important components: error-tolerant representations and explicit error-correction strategies. In addition, exploration can be helpful, but should be optional. This can be implemented through the use of exploration scripts, as described above.

Our mapping system, as we have described, is a pas-

sive mapping system designed for indoor environments. In the future, we want to extend this work to other types of environments, such as city streets or forests. At present, though, this work is directly applicable to some real-world mapping tasks, such as those involved in inter-office delivery or some search-and-rescue tasks. Implementation of complete systems that could be deployed for such tasks is not yet feasible; it awaits other developments in effective perception and robust action.

References

[1] G. Alefeld and J. Hertzberger. *Introduction to Interval Computation.* Academic Press, New York, NY, 1983.

[2] Sami Atiya and Greg Hager. Real-time vision-based robot localization. In *Proc. Int'l Conf. on Robotics and Automation*, 1991.

[3] Nicholas Ayache and Olivier D. Faugeras. Maintaining representations of the environment of a mobile robot. *IEEE Trans. on Robotics and Automation*, 5(6), 1989.

[4] Kenneth Basye, Tom Dean, and Jeffrey S. Vitter. Coping with uncertainty in map learning. Technical Report CS-89-27, Brown University Department of Computer Science, June 1989.

[5] David J. Braunegg. *MARVEL: A System for Recognizing World Locations with Stereo Vision.* PhD thesis, MIT, 1990.

[6] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proc. National Conference on Artificial Intelligence*, pages 183–188, 1992.

[7] Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, 1987.

[8] Sean P. Engelson. Active place recognition using image signatures. In *Proceedings of SPIE Symposium on Intelligent Robotic Systems, Sensor Fusion V*, 1992.

[9] Sean P. Engelson. *Passive Map Learning for Mobile Robots.* PhD thesis, Department of Computer Science, Yale University, 1994. (forthcoming).

[10] Sean P. Engelson and Niklas Bertani. ARS MAGNA: The abstract robot simulator manual. Technical Report YALEU/DCS/TR-928, Yale University Department of Computer Science, 1992.

[11] Sean P. Engelson and Drew McDermott. Passive robot map building with exploration scripts. Technical Report YALEU/DCS/TR-898, Yale University Department of Computer Science, March 1992.

[12] David J. Kriegman. *Object Classes and Image Contours in Model-Based Vision.* PhD thesis, Stanford University, 1989.

[13] Benjamin Kuipers. Modeling spatial knowledge. *Cognitive Science*, 2:129–153, 1978.

[14] Benjamin Kuipers and Yung-Tai Byun. A robust qualitative method for robot spatial reasoning. In *Proc. National Conference on Artificial Intelligence*, pages 774–779, 1988.

[15] T. S. Levitt, D. T. Lawton, D. M. Chelberg, and P. C. Nelson. Qualitative landmark-based path planning and following. In *Proc. National Conference on Artificial Intelligence*, Seattle, Washington, 1987.

[16] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. Technical Report 1228, MIT Artificial Intelligence Laboratory, 1990.

[17] David P. Miller and Marc G. Slack. Global symbolic maps from local navigation. In *Proceedings of IJCAI-91*, pages 750–755, 1991.

[18] Franco Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 2nd edition, 1988.

[19] Karen B. Sarachik. Visual navigation: Constructing and utilizing simple maps of an indoor environment. Technical Report 1113, MIT Artificial Intelligence Laboratory, 1989.

[20] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Proceedings of the Second Workshop on Uncertainty in Artificial Intelligence*, Philadelphia, PA, 1986.

[21] C.J. Taylor and David Kriegman. Structure and motion from line segments in multiple images. In *Proc. Int'l Conf. on Robotics and Automation*, May 1992.

[22] Wai K. Yeap. Towards a computational theory of cognitive maps. *Artificial Intelligence*, 34(3), April 1988.

EXTENSIBILITY IN LOCAL SENSOR BASED PLANNING FOR HYPER-REDUNDANT MANIPULATORS (ROBOT SNAKES)

Howie Choset Joel Burdick
Dept. of Mechanical Engineering
Mail Code 104-44, CALTECH, Pasadena, CA 91125

ABSTRACT: This paper extends a *local* sensor based planning method for hyper-redundant robot mechanisms. In a previous paper, sensor feedback control methods are considered. A highly localized sensor feedback method for hyper-redundand manipulators is termed, *partial shape modification* (PSM). A PSM utilizes a mechanism's hyper-redundancy to enable both local obstacle avoidance and end-effector placement in real time. This paper considers the situation in which the limits of a PSM is violated. In other words, what does the robot do when it can not only locally adapt to the environment. Local sensor based planning has been implemented on a thirty degree of freedom hyper-redundant manipulator which has eleven ultrasonic distance measurement sensors and twenty infrared proximity sensors. The robot is controlled by a real time control computer which communicates with sensors through an innovative sensor bus architecture. Experimental results obtained using this test bed show the efficacy of the proposed method.

1. Introduction

This paper extends experimental results from [TCB] in the area of *local sensor based planning* for hyper-redundant robot manipulators. Recall from [ChB90b] that a "hyper-redundant" manipulator is a kinematically redundant manipulator in which the degree of redundancy is very large or infinite. Such robots are analogous in morphology to tentacles, an elephant's trunk, a monkey's tail or a snake. "Sensor based planning" incorporates sensory information into some stage of a robotic motion planning, whether it be navigation, locomotion, grasping, etc. "Local Sensor Based Planning" fine tunes a robot's plan, based on sensor information. Local sensor based planning is useful when: (1) the robot only has a coarse knowledge of the world because of limited memory; (2) the robot's world model contains inaccuracies; and (3) the world is subject to unexpected occurrences or rapidly changing situations. These situations can be overcome with local sensor based planning strategies.

Due to their many degrees of freedom, hyper-redundant robots are potentially superior for operations in highly constrained and unusual environments encountered in applications such as inspection of nuclear reactor cores, chemical sampling of buried toxic waste, and medical endoscopy. Hyper-redundant robots can also be used as tentacle-like grasping devices for capturing and manipulating floating satellites [ChB90c] or to enable complex "whole arm manipulation." Mobile hyper-redundant robots also offer novel means for locomotion [ChB91a, ChB93a, ChB93b, ChB93c] in complex environments.

The above mentioned applications are characterized by environments which are difficult to precisely model and which are time varying. Thus, local sensor-based motion planning schemes are vital to the realistic deployment of hyper-redundant robots in these applications. While hyper-redundant robots have many advantages for the above described applications, they have one disadvantage. Since hyper-redundant manipulators have a large number of joints or actuators, small joint displacement errors can accumulate to reasonably large errors in the position of the tip relative to the base. Thus, the effective accuracy of hyper-redundant robots could be improved by distributing sensors along their length and employing sensor based planning schemes.

Thus, local sensor based planning can be used to: (1) account for spatial uncertainty or inaccuracies in the world model used by a "global" planner to construct a robot plan; (2) increase the effective accuracy of a hyper-redundant robot mechanism; and (3) locally adapt to rapid environmental variations, such as moving obstacles, that can not be easily or rapidly handled by a global planner.

The local sensor based planning algorithm of hyper-redundant manipulators is based on the analysis found in [ChB90b, ChB91a, ChB92a, ChB92c, Ch]. This work has been demonstrated on an actual extensible 30 degree-of-freedom hyper-redundant robot system. A hyper-redundant manipulator which can vary its length, within the limitations of its actuators, is termed "extensible." A more detailed account of this mechanism and its capabilities can be found in [ChB92b].

Robotic motion planning has been an important area of research. Since the introduction of configuration space methods [LoWes], several other theories have been published, some of which are summarized in [Sh,Lat]. However, these methods plan from a perfect model of the world, which is normally unavailable to a real robot. More recently, methods have been developed in which the robot explores the environment to gather information for the planning process [CaLi]. These approaches assume that the sensors provide perfect information about the environment. There has been little work devoted explicitly to motion planning for robot snakes. One approach is based on the construction of *tunnels* through the ob-

Copyright 1994 by the American Institute of Aeronautics and Atronautics, Inc. All rights reserved.

stacle field, through which the manipulator "slithers" [ChB90a, Ch]. In another work, sensor based planning for highly redundant robots is based on a *tactrix* [ReLu]. However, this work assumes that there are perfect sensors on the robot; nor has it been implemented on a real robot. Hirose [HiU] implemented an "active cord" mechanism, which used tactile sensors to guide its motion. A previous paper [TCB] presents preliminary strategies for local sensor based planning, which are implementable in real time, can employ a variety of sensors, and exploit the benefits of hyper-redundancy.

In this paper, a local sensor based planning strategy for hyper-redundant manipulators is extended so it can be more accommodating. The local sensor based planner in [TCB] did not consider its own limitations. There, the hyper-redundant manipulator can only locally adapt to a changing environment over a fixed length of the robot. For a fixed length of robot, a hyper-redundant manipulator uses its extensibility so it can locally avoid obstacles. However, the robot can only deform until its joint limits are met, in which case the hyper-redundant manipulator can no longer adapt. In other words, the robot used up all of its extensibility over the fixed length. The longer this fixed length, the more the robot can deform. However, the longer the length, the less local the response, which is undesireable. In the new algorithm, the length of the deforming part of the robot is variable, therefore enhancing its ability to locally adapt. In effect, the manipulator uses more of its extensibility from other parts of the robot to locally avoid objects. Experiments demonstrate that local sensor based planning is not only useful, but also implementable in real time with very reasonable computing power and simple sensors.

The structure of this paper is as follows. Section 2 reviews the basic framework of hyper-redundant manipulator kinematics which is based on "backbone curves." The backbone curve and its deformation is the basis for the algorithms of Section 3. We primarily consider algorithms for planar mechanisms, as the experimental verification of these ideas was performed on a planar robot. Many of these algorithms can be extended to the spatial case. Section 4 describes the experimental setup, while Section 5 described the actual results of these experiments.

2. Background

This section reviews a hyper-redundant robot kinematic analysis framework that forms the basis of this work. Recall from [ChB90b] that we assume that (regardless of mechanical implementation) the important macroscopic features of a hyper-redundant robot can be captured by a *backbone curve*. A backbone curve parametrization and an associated set of reference frames which evolve along the curve are collectively called the *backbone reference set*. In this paradigm, inverse kinematics and task planning reduces to the determination of the proper time varying behavior of the backbone reference set [ChB90b]. Similarly, local sensor based planning is equivalent in this approach to modification of the backbone curve shape in order to accommodate impinging obstacles.

In [ChB92c], many techniques are introduced for parametrizing the backbone curve. In this paper, we will assume that the Cartesian position of points on a backbone curve can be parametrized in the form:

$$\vec{x}(s,t) = \int_0^s l(\sigma,t)\ \vec{u}(\sigma,t)d\sigma \qquad (2.1)$$

where $s \in [0,1]$ is a parameter measuring distance along the backbone curve at time t. The *backbone curve base* is the point $s = 0$. $\vec{x}(s,t)$ is a vector from the backbone curve base to point s. By convention, $\vec{x}(0,t) = 0$. $\vec{u}(s,t)$ is the unit tangent vector to the curve at s. $l(s,t)$ is the length of the curve tangent and assumes the general form:

$$l(s,t) = 1 + \epsilon(s,t) > 0. \qquad (2.2)$$

$\epsilon(s,t)$ is the *local extensibility* of the manipulator, which expresses how the backbone curve locally expands or contracts relative to a fixed reference state. We show later on that the robot needs this extensibility in order to locally avoid obstacles.

The parametrization of Eq. (2.1) has the following interpretation. The backbone curve is "grown" from the base by propagating the curve forward along the tangent vector, which is varying its direction according to $\vec{u}(s,t)$ and varying its magnitude (or 'growth-rate') according to $l(s,t)$.

Our experiments have been performed on a device with planar geometry. In the planar case, the backbone curve is the locus of points:

$$\vec{x}(s,t) = [x_1(s,t),\ x_2(s,t)]^T$$

where

$$x_1(s,t) = \int_0^s l(\sigma,t)\sin\theta(\sigma,t)d\sigma \qquad (2.3)$$

$$x_2(s,t) = \int_0^s l(\sigma,t)\cos\theta(\sigma,t)d\sigma. \qquad (2.4)$$

$\theta(s,t)$ is the angle, measured clockwise, which the tangent to the curve at s makes with the x_2-axis at time t. By convention (2.4), $\theta(0) = 0$, and $x_1(0) = x_2(0) = 0$. By comparing equations (2.1) with equations (2.3) and (2.4), it easy to see that $\vec{u}(s,t) = [\sin\theta(s,t),\ \cos\theta(s,t)]^T$ in the planar case. $l(s)$ and $\theta(s)$ are termed "shape functions," as they control they shape of the backbone curve through the forward kinematic relations (2.3) and (2.4).

Within the context of this modeling technique, the inverse kinematic problem, or "hyper-redundancy resolution" problem, reduces to the determination of the time varying behavior of backbone curve shape

functions that satisfies task requirements. Different hyper-redundancy resolution techniques can be found in [ChB90a, ChB91a, ChB92a, ChB92c, Ch]. In one approach, which is relevant to the algorithm of Section 3, the backbone curve shape functions are restricted to a "modal form"

$$\theta(s,t) = \sum_{i=1}^{N_\theta} a_i(t)\,\Phi_i(s) \qquad l(s,t) = \sum_{i=N_\theta+1}^{N_l} a_i(t)\,\Phi_i(s) \tag{2.5}$$

where $\Phi_i(s)$ is a "mode function," and $a_i(t)$ is the associated "modal participation factor." $N = N_\theta + N_l$ is the total number of modes, which must equal or exceeds the number of task constraints. Hyper-redundancy is resolved in the modal approach by constraining the backbone curve to N effective DOF. The $\{\Phi_i\}$ are predetermined functions chosen by the programmer, and can often be selected to incorporate physical characteristics of the task [Ch]. Thus, the backbone curve geometry becomes solely a function of the $\{a_i\}$. The inverse kinematics problem reduces to finding the $\{a_i\}$ which satisfy task constraints. In [ChB91a, ChB92c], closed form solutions are given for several choices of mode functions.

A continuous backbone curve inverse kinematic solution is used to determine the actuator displacements of a continuous morphology robot such as one constructed from pneumatic actuator bundles. For discretely segmented morphologies, such as the prototype described in this paper in Section 4, the continuous curve solution can be used, via a "fitting" process, to compute the actuator displacements which cause the manipulator to exactly assume or closely approximate the continuous backbone curve model. The fitting techniques which are used in subsequent examples are reviewed in [ChB91a, ChB92c].

3. Local Sensor Based Planning Algorithm

Local Sensor-Based Planning (LSBP) assumes that a backbone curve is somehow determined by a high level global planning process. The backbone curve shape is then modified in response to sensory information. LSBP does not use a model of the environment, and is intended for rapid response to environment changes. In order to describe the local sensor based planning strategy, a sensor model must be described.

3.1. Sensor Models

The algorithm described below uses a very simple sensor model. We assume that the sensors are rigidly attached to the backbone curve at a fixed point. That is, they move with the backbone curve, and their orientation is a function of the backbone curve tangent at the point of attachment. The sensors are assumed to measure, along a fixed direction termed the *sensor measurement axis*, the distance to a nearby obstacle. The sensor measurement axis is a function of the sensor and the backbone curve geometry (See Fig. 1). Our sensors *do not* measure the distance to the point on the obstacle which is nearest to the backbone curve. Rather, they measure the distance which would actually be computed by realistic sensors. This simple model is representative of the infrared and ultrasonic sensors discussed in Section 4. In addition, there is often some directional ambiguity due to the finite width of a typical sensor's beam pattern. We assume that the sensor measurement axis is the centerline of the beam pattern. The distance measurement returned by the sensor is the nearest point of the obstacle lying within beam pattern cone. Since it is impossible to resolve the angular ambiguity, we assume that nearest point of the obstacle lies along the beam pattern centerline.

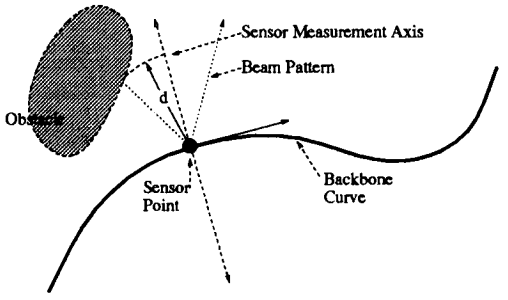


Figure 1: Simplified Distance Measurement Sensor Model

3.2. Partial Shape Modification Control

This section describes a PSM planning strategy in which the backbone curve is approximated by a large number, n_d, of discrete endpoints. The sensors are assumed to be rigidly attached at points along the discretized backbone curve. There are typically many approximating segments between adjacent sensor attachments. When a sensor detects the presence of an obstacle, the backbone curve shape locally deforms in a region around the sensor. In our simulations and experiments, $n_d \sim 100$, and there were about 10 discrete points between sensor points.

The actual response, a displacement of the approximating points, is determined by a *local sensor response function* (LSRF), which is assumed to be a discrete unimodal function. A unimodal function is one which has one local maximum, the global maximum, over its domain. The response function is "added" to the current backbone curve, locally drawing it away from an obstacle. In Figure 2, a triangle LSRF is added to a straight backbone curve, deforming the backbone curve away from an sufficiently close obstacle.

Since the robot only detects the obstacle at a sensor point, the reaction to the obstacle should be greatest at the sensor point, and should monotomically decrease at points away from the sensor point. Therefore, the sensor point is the center of this unimodal

function, and is assumed to be farthest away from the obstacle.

The LSRF should also look like the beam pattern of the sensor associated with the sensor point. Typically, beam patterns have a central lobe along the sensor axis, in which the obstacle likely lies. In the experimental setup, the spatial resolution of the robot's actuators is much lower than the azimuth resolution of the sensors on the robot. Therefore, a simple triangle (or cone) is a sufficient approximation to the main lobe of the beam pattern, and thus, a reasonable choice for a LRSF. Later on, it is shown that a triangle response function leads to a trivial and efficient solution to LSBP for planar hyper-redundant manipulators. So, the example displayed in Figure 2 is a good example of a LSRF.

The half width of the LSRF is slightly larger than the distance between two adjacent sensors on the backbone curve. This way, if two adjacent sensors detect the same obstacle, their cumulative response function is still unimodal.

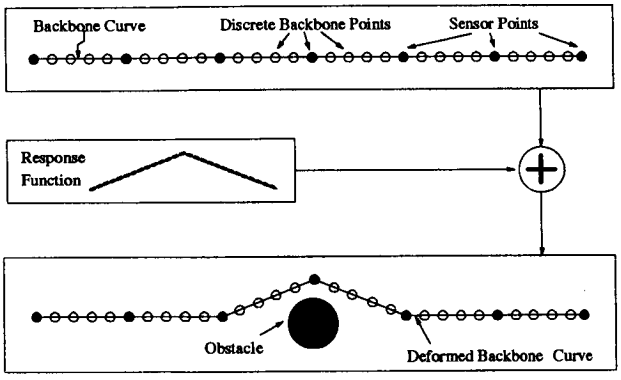


Figure 2: Backbone, Response Function, and Deformed Backbone

In this approximation method, the position of the discrete segment endpoints can be approximated by the discretization of the continuous forward kinematics integral (Eq. (2.1)):

$$\vec{p}(s_i,t) = \sum_{k=0}^{k=i} l(s_k,t)\vec{u}(s_k,t) \qquad (3.2.1)$$

$l(s)$ and $\vec{u}(s)$ are continuous shape functions which are specified by a global planner. They need not assume a modal form. Also, an endpoint may or may not coincide with a sensor point.

A small differential change in $\vec{p}(s_i)$ is:

$$\delta\vec{p}(s_i,t) = \sum_{k=0}^{k=i} \delta l(s_k,t)\vec{u}(s_k,t) + l(s_k,t)\delta\vec{u}(s_k,t) \qquad (3.2.2)$$

where $s_k = \frac{k}{n_d}$, where n_d is the number of discrete points along the back bone curve. $\delta\vec{u}$ is a local change in the backbone curve tangent direction, while $\delta\vec{l}$ represents a local stretch.

The goal of this PSM method is to compute the local perturbations, $\delta\vec{u}$ and $\delta\vec{l}$, which deform the backbone curve away from obstacles. The changes in backbone curve tangent and stretch are determined from $\delta\vec{p}$, which in turn is determined by the LSRFs. The modified backbone curve shape is then used by the fitting algorithms to determine the appropriate actuator displacements.

Assume that at some initial time, a global planner specifies a backbone curve shape. Thus, $\delta\vec{p} = 0$ initially. For each sensor point along the backbone curve that detects an obstacle within its response envelope, a discrete unimodal LSRF is added to (or subtracted from, depending upon from which direction an obstacle appears to be) δp, the vector which contains the prescribed changes to the backbone curve. Setting $\delta p(s_n,t) = 0$, guarantees that, within the limits of the discretization approximation, the end effector position will not change. Setting $\delta p(s_n,t) = 0$, $\delta p(s_{n-1},t) = 0$, and $\delta\vec{u}(s_n,t) = \vec{0}$ guarantees that the end effector position and orientation will not change. The new backbone curve can be computed after $\delta\vec{u}$ and $\delta\vec{l}$ are determined from (3.2.1).

In the case of a planar backbone curve, (3.2.2) can be written in matrix form:

$$\begin{pmatrix} \delta p_x^1 \\ \delta p_y^1 \\ \delta p_x^2 \\ \delta p_y^2 \\ \vdots \\ \delta p_x^{n-1} \\ \delta p_y^{n-1} \\ \delta p_x^n \\ \delta p_y^n \end{pmatrix} =$$

$$\begin{pmatrix} l_1 & 0 & 0 & 0 & \dots & 0 & 0 & u_x^1 & 0 & 0 & \dots & 0 \\ 0 & l_1 & 0 & 0 & \dots & 0 & 0 & u_y^1 & 0 & 0 & \dots & 0 \\ l_1 & 0 & l_2 & 0 & \dots & 0 & 0 & u_x^1 & u_x^2 & 0 & \dots & 0 \\ 0 & l_1 & 0 & l_2 & \dots & 0 & 0 & u_x^1 & u_y^2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ l_1 & 0 & l_2 & 0 & \dots & 0 & 0 & u_x^1 & u_x^2 & u_x^3 & \dots & 0 \\ 0 & l_1 & 0 & l_2 & \dots & 0 & 0 & u_y^1 & u_y^2 & u_y^3 & \dots & 0 \\ l_1 & 0 & l_2 & 0 & \dots & l_n & 0 & u_x^1 & u_x^2 & u_x^3 & \dots & u_x^n \\ 0 & l_1 & 0 & l_2 & \dots & 0 & l_n & u_y^1 & u_y^2 & u_y^3 & \dots & u_y^n \end{pmatrix} \begin{pmatrix} \delta u_x^1 \\ \delta u_y^1 \\ \delta u_x^2 \\ \delta u_y^2 \\ \vdots \\ \delta u_x^n \\ \delta u_y^n \\ \delta l_1 \\ \delta l_2 \\ \delta l_3 \\ \vdots \\ \delta l_n \end{pmatrix} \qquad (3.2.3)$$

where $\delta\vec{p}^i = \delta\vec{p}(s_i,t)$, $\delta\vec{u}^i = \delta\vec{u}(s_i,t)$, $\delta\vec{l}_i = \delta\vec{l}(s_i,t)$, $\vec{l}_i = \vec{l}(s_i)$, and $\vec{u}^i = \vec{u}(s_i,t)$. The x and y components of $\delta\vec{p}(s_i,t)$ are respectively denoted δp_x^i and δp_y^i. Similary, the x,y components of $\delta\vec{u}(s_i,t)$ are δu_x^i and δu_y^i. $\delta\vec{p}$ and $\delta\vec{u}$ each have $2n$ elements, and $\delta\vec{l}$ has n elements.

For given $\delta\vec{p}$, there is not a unique solution to Eq. (3.2.3). A simplified (and unique) solution for

(3.2.3) is obtained by setting $l(s_i,t) = 1\ \forall_{1\leq i\leq n}$ (i.e $\delta l(s_i,t) = 0$). Although all of the $l(s_i,t) = 1$, the snake can still use its extensibility to avoid obstacles because $\|\vec{u}^i + \delta\vec{u}^i\| \neq 1$. In other words, the length of the tangent vectors are no longer constrained to have unit length in this approximation unless additional restrictions are employed. After setting $\delta l(s_i,t) = 0$ and enforcing the end effector constraints,(3.2.3) becomes:

$$\begin{pmatrix} \delta p_x^1 \\ \delta p_y^1 \\ \delta p_x^2 \\ \delta p_y^2 \\ \vdots \\ \delta p_x^{n-2} \\ \delta p_y^{n-2} \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 1 & 0 & \dots & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & \dots & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & \dots & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & \dots & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \delta u_x^1 \\ \delta u_y^1 \\ \delta u_x^2 \\ \delta u_y^2 \\ \vdots \\ \delta u_x^{n-2} \\ \delta u_y^{n-2} \\ \delta u_x^{n-1} \\ \delta u_y^{n-1} \end{pmatrix} \tag{3.2.4}$$

which has a simple, obvious and easily computed solution to (3.2.4).

$$u_x^i = p_x^i - p_x^{i-1} \tag{3.2.5}$$

$$u_y^i = p_y^i - p_y^{i-1} \tag{3.2.6}$$

The discretized backbone curve is then used, via a fitting procedure, to compute the local actuator displacements which implement the desired deformation. Figure 3 displays a PSM deformation of a 30 DOF variable geometry truss manipulator (kinematically identical to the real system described in Section 4) in response to an impinging obstacle. The backbone curve is approximated by 100 segments. The manipulator is originally in a straight configuration, which locally deforms to avoid a simulated obstacle in Figure 4. In this simulation, the response function is shaped like a triangle.

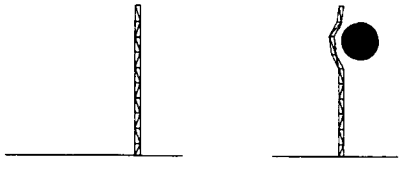


Figure 3: Original **Figure 4:** Resulting

3.3. Extended PSM

Due to mechanical limitations of the robot, such as joint limits, a real hyper-redundant manipulator has a limited amount of local extensibility. That is, $\|\epsilon\| < \Upsilon$ where Υ is the limit of local extensibility of the manipulator. This means that at any point, there is a fixed range over which the backbone can expand or contract relative to a fixed reference state.

The original PSM assumes that there is infinite local extensibility, which is unrealistic for actual robots. In the example in figure 5, an object becomes unacceptably close to the robot, which locally moves away from the object using an LSRF. However, the object continues to move towards the already deformed robot, which wants to move further away locally from the object, using the same LSRF. Although a backbone curve is determined in this situation, it is not likely that a real mechanism can fit this curve because this backbone requires a lot of local extensibility from the manipulator. In this case, the $\|\epsilon\| < \Upsilon$ constraint was violated because $\|\delta\vec{u}\,(s_i,t)\| > \Upsilon$.

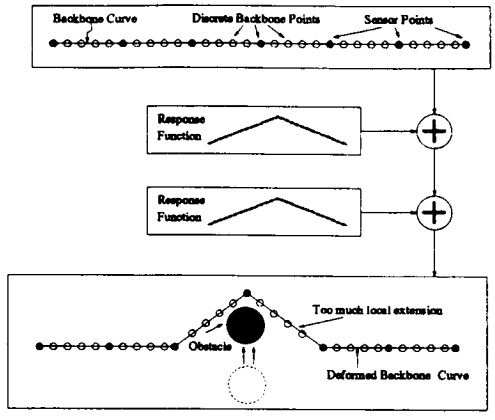


Figure 5: Same Response

So, a new LSRF has to be used which utilizes the extensibility of neighboring regions on the backbone curve, while not using any local extensibility from the already deformed section (i.e. maintaining the constraint $\|\epsilon\| < \Upsilon$). This is called "sucking" extensibility from other parts of the robot snake. In figure 6, the local extensibility limit is not exceeded, and the robot is still able to accommodate for the object's displacement.

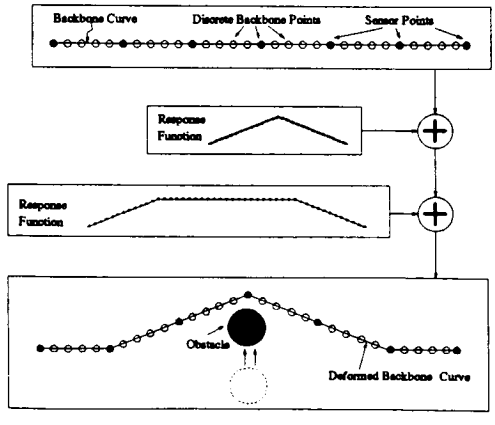


Figure 6: Modified Response

4. Experimental Setup

To prove the feasibility of the proposed algorithms, a distributed sensor system was developed for the 30 degree-of-freedom hyper-redundant robot system described in [ChB92b]. Figure 7 shows the structure of this testbed. This section describes the test bed structure in detail.

4.1. Hyper redundant manipulator and control system

The hyper-redundant manipulator is a modular Variable Geometry Truss design [Ch]. The 30 degree-of-freedom (DOF) planar robot consists of ten modules (also called bays) of 3 DOF each (Fig. 7). Each DOF consists of a D.C. servo motor which drives a lead screw. Each lead screw is instrumented with a linear potentiometer. The real time system controller is based on a VME-bus multiprocessing computer, currently consisting of two Heurikon (68030 and 68020) single board processors, and the VxWorks real time operating system. One processor is dedicated to the closed loop feedback control of the actuator positions. The other processor is dedicated to processing of sensor data and real time computation of the PSM algorithms.

To enable flexible, modular, and easily expandable experimentation with sensor based planning, a novel 34 wire "Sensor Bus" architecture was developed for the sensor system. One end of the sensor bus is connected to the PSM processor via a parallel port. The sensor bus consists of an eight bit outgoing data path, a four bit status line, a two bit strobe and one interrupt request line. The data path and the two strobe lines enable the CPU to access up to 256 sensors and to send eight bits of information to the sensor peripherals for possible sensor control purposes. The interrupt request line is connected to the hardware counter on the CPU board so that accurate timing measurements can be made in real time.

Sensors can be added to the system via "Sensor Interface Modules." This module decodes the sensor bus address and generates signals to control sensors. Up to two ultrasonic sensor modules and six sets of sensors which produce data with 4 bit (or less) quantization can be controlled. Currently, only four infrared sensors per board are present, though up to eight infrared sensors and eight mechanical switchs can be directly connected. The sensor interface module circuitry is mounted on a printed circuit board which is 15cm by 12cm in size. Fig. 8 shows a photograph of the sensor interface module. The sensor bus is physically connected in the bottom of the module in a daisy-chain fashion. In the figure, two ultrasonic transducers are shown above the module. Also the infrared proximity sensor is shown on the side of the module.

4.2. Sensors

Currently, the robot has two types of sensors: infrared (IR) and ultrasonic (US). There are five sets of two US sensors. Each set is rigidly attached to alternate bays so that each sensor points outward from the backbone curve (or the centerline of the mechanism). An additional US sensor will be mounted at the front of the mechanism, pointing forward. Twenty IR sensors, again two at each the ten bays facing outward, are currently mounted on the snake. In the near future, twenty-four additional IR sensors will be added, two more at each bay, and four in front. Fig. 9 schematically shows how these sensors are distributed throughout the mechanism.

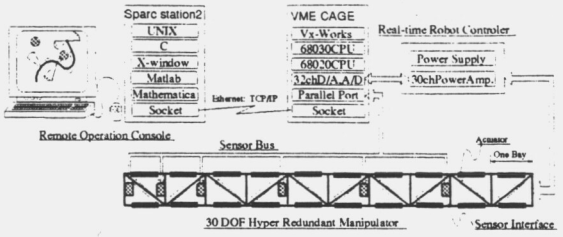


Figure 7: Hyper Redundant Robot Test Bed

Figure 8: Sensor Interface Module

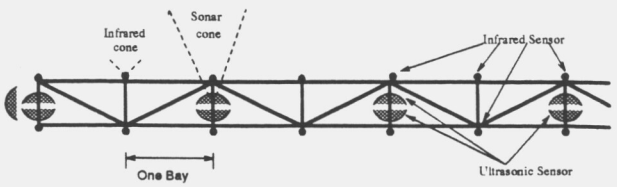


Figure 9: Sensor Arrangement

Electrostatic type ultrasonic sound transducers determine distance by measuring the time of flight of the ultrasound pulse leaving the transducer, bouncing off an object and returning to the sensor. A 50kHz sonar wave burst is transmitted when the sonar-ranging module is triggered [Ci]. The ranging module output is connected to the sensor bus interrupt request line. The echo return time is computed by the CPU hardware counter. Since the sixteen bit resolution distance measurement result is read from the hardware counter, no result is ever sent through the sensor bus. This design greatly simplifies the hardware required.

The US sensors are activated sequentially (at 16 millisecond intervals) to prevent interference between sensors. These sensors are calibrated to measure distances ranging from 10cm to 2.5m, with a 2% accuracy. There is about twenty degrees of conical ambiguity for direction, because of the transmitting beam pattern of the transducer [Ci]. In this work, it is assumed that the obstacle lies along the cone's centerline, which is locally normal to the backbone curve at the point of sensor attachment.

The IR sensors yield binary proximity information– i.e., the presence or absence of the obstacle in some pre-set range. An infrared LED emits modulated infrared light, and if an obstacle is near the robot, the IR sensor will detect the reflected light. The range of the IR system can be adjusted by setting potentiometers on the sensor boards. Currently, the IR system is set up to detect the presence of obstacles up to four inches away from the robot. Like the US, the location of an obstacle is not precisely known, but lies somewhere in a cone emanating from the IR sensor.

Each sensor has its own advantage. The IR sensors have a very fast response and can be sampled at extremely high rates. They are thus suitable for the PSM system. The US sensors provide proportional obstacle distance, rather than binary proximity information. They are thus more useful for accurate planning. However, since the US sensors are sequentially polled to prevent interference, the minimum sampling period is 176 milliseconds. The IR sensors are sequentially polled in a similar fashion, but at a significantly higher rate. To maximize the use of both types of sensors, the sensor interface module is designed to operate both US and IR sensors simultaneously in different intervals.

4.3. Remote Operation Console

The real time computers are connected to Sun workstations via the ethernet. Via software sockets, information can be transferred through the ethernet between the real time computer running VxWorks and the Sun workstations running Unix. C programs and many software packages, such as Matlab, are able to directly communicate with the real time computers via the sockets. Therefore, these programs can control the snake. The FSM and higher levels of control are implemented on the SUN workstation.

Experimental robot control programs are developed

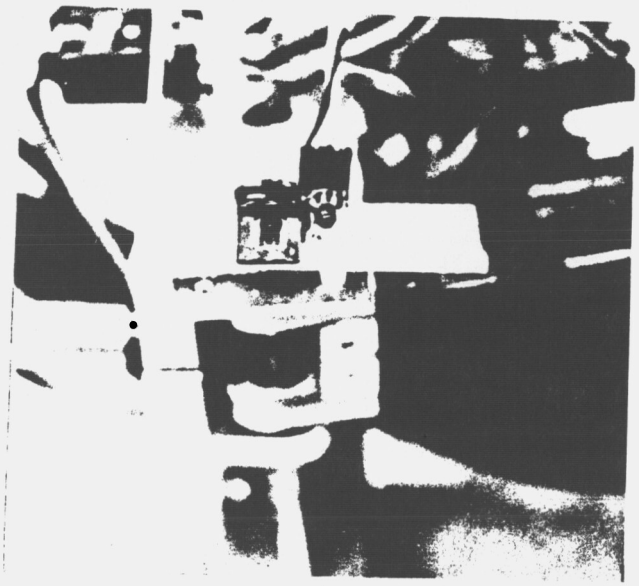


Figure 10: Infrared Sensor

in a combination of C and Matlab. Via an X-Window interface, these programs graphically display and continually update the robot's configuration and sensor measurements. Fig.11 shows the X-Window operation console window. In addition, many motion planning and sensing commands can be executed using a graphical menu interface. End-effector via points of a hyper-redundant trajectory can be specified by a mouse, and the trajectory is then executed by the real-time system.

In addition to graphically depicting the current configuration of the manipulator, this system displays US and IR sensor measurements. The solid cones emanating from the manipulator represent US sensor data. In this representation scheme, the closet point to an obstacle in the sensor beam pattern lies somewhere on the distal arc of the cone. The dashed arcs much closer to the mechanism indicate that the IR sensors have detected nearby obstacles at these locations.

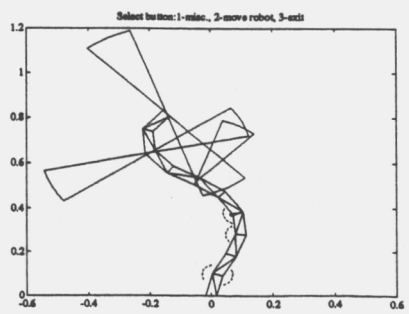


Figure 11: Operation Console

5. Results

The PSM algorithms described in Section 3.2 and 3.3 have been implemented on our hyper-redundant robot test-bed. Photographs of two experiments are shown below. In the first experiment, the backbone curve, dictated by some high level planner, was a straight line. Two obstacles were moved into an unacceptably close proximity (about 10cm or 4in) to the mechanism, and the manipulator locally deformed away from each obstacle while maintaining constant end-effector position. Truthfully, the end-effector was displaced slightly from its original position (less than a 1 inch displacement over a distance of $\sim$16 feet). The current implementation of the discrete approximation algorithm employs only IR sensors, because there are many more IR sensor distributed along the snake. See figure 12.

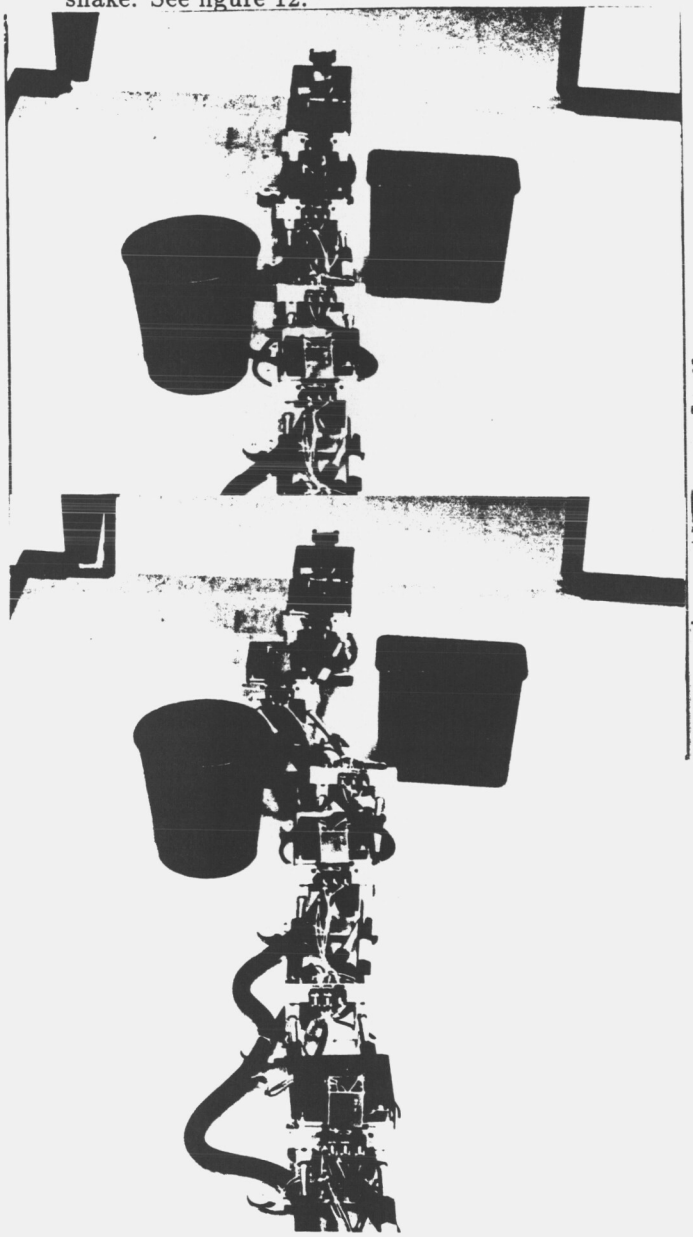


Figure 12: First Experiment

In the next experiment, again, the backbone curves starts off as a straight line. See figure 13. One obstacle was moved unacceptably close to the robot which resulted in the mechanism moving, as it did in the first experiment. See figure 14. Then, the object was moved sufficiently close to the deformed robot, passing through the original backbone curve, and the manipulator still deformed away. See figure 15. Such a large local deformation would not have been possible with the original PSM.

Th second experiment showed the local shape modification capability of the new PSM algorithm proposed in this paper. In real time, the old PSM reliably works, when the actuators in the section of the robot that is deforming are not near their joint limits. In such a case, the new PSM is the same as the old PSM. As actuators' limits are approached, local deformation may become infeasible, and this is where the new PSM becomes useful. The new planner uses extensibility from neighboring actuator displacements along the manipulator so, the robot can still locally deform. However, once all the actuators reach there limit, i.e. all the extensibility is used up, the robot has to report to a higher level planner in order to accommodate for all the constraints in the environment. This ability is currently being implemented in our system.

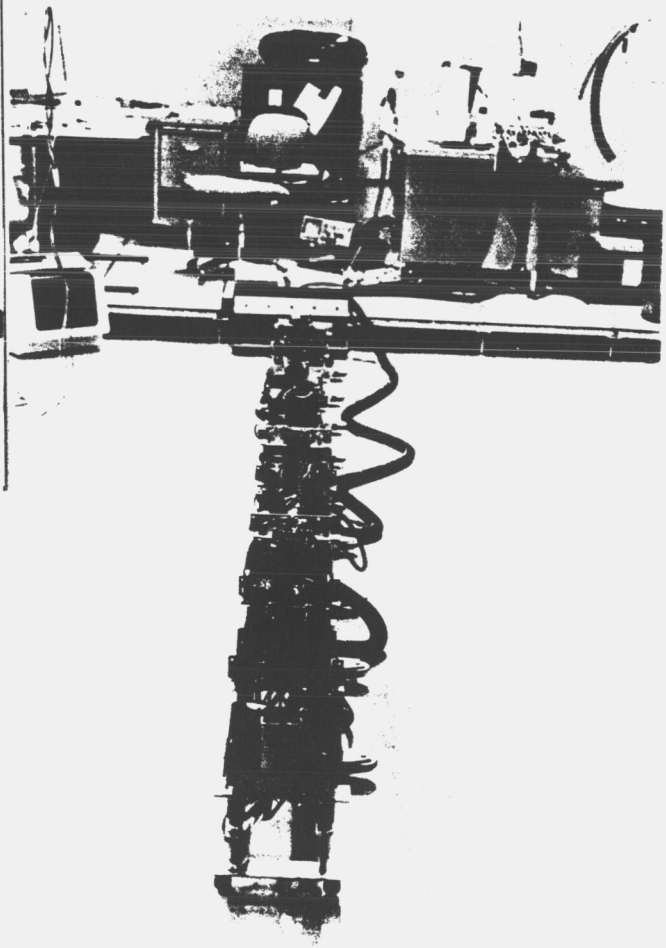


Figure 13: Second Experiment

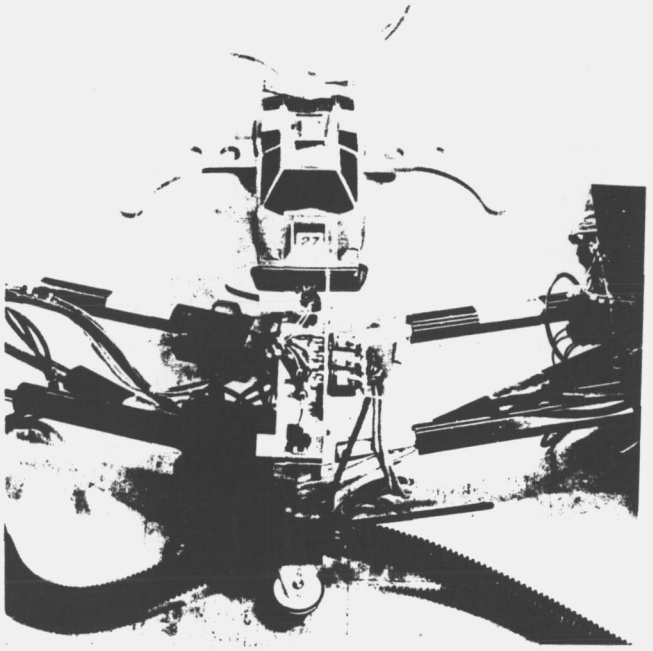


Figure 14: Intermediate

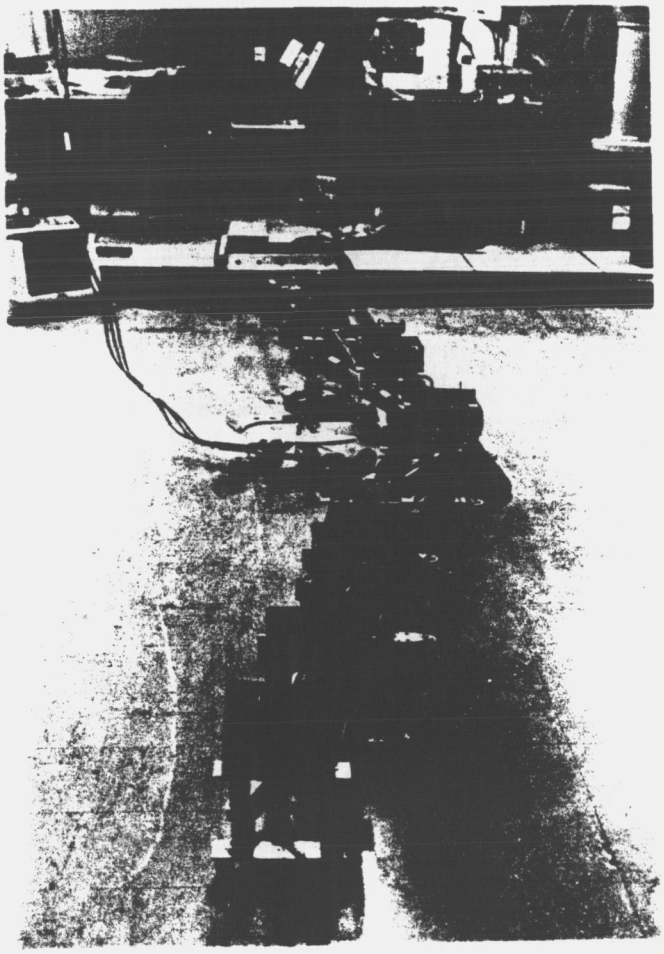


Figure 15: Final

6. Conclusion

In this paper, a local sensor based planning method for hyper-redundant robots is extended. This method is based on a backbone curve kinematic framework. In the previous work, the limit of local deformation was limited to the extensibility over a fixed portion of the robot. In this paper, in order to better compensate for objects penetrating the backbone curve, extensibility was used over a variable portion of the robot.

This method was implemented on an actual 30 DOF hyper-redundant manipulator test bed. An innovative sensor bus architecture and a graphical programming and display interface were reviewed. Experiments using this system showed the applicability and effectiveness of the proposed method to real hyper-redundant manipulators. A reasonable amount of computer power was required for real-time implementation of these algorithms.

As suggested in the previous section, we are currently working to improve the communication between low level planners and a high level planner so that the robot can better interpret and react to exceptional conditions indicated by the PSM level. In addition, we intend to develop better sensor function methods which properly combine ultrasonic and infrared sensor readings from adjacent sensors. The highly distributed nature of sensors on a hyper-redundant mechanism also point to the need for new theories on deploying and using massively redundant sensor arrays. Finally, future work will focus on using sensor data for higher level, i.e. global, hyper-redundant robotic planning.

Acknowledgements: This work has been supported by National Science Foundation Presidential Young Investigator Grant MSS-9157843, and by the office of Naval Research Young Investigator Award N00014-92-J-1920 and N00014-93-1-0782. The authors would also like to thank Mr. Nobuaki Takanashi of the NEC Corporation.

7. References

[CaLi] Canny J. F., Lin, M. C. "An Opportunistic Global Path Planner," *Proc. IEEE Int. Conf. on Robotics and Automation*, Cincinnati, Ohio, May 14-17, 1990, pp. 1554-1559.

[Ch] Chirikjian, G. "Theory and Applications of Hyper-Redundant Robotic Manipulators," PhD Thesis, California Institute of Technology, Pasadena, Ca, 1992.

[ChB90a] Chirikjian, G.S., Burdick, J.W., "An Obstacle Avoidance Algorithm for Hyper-Redundant Manipulators," *Proc. IEEE Int. Conf. on Robotics and Automation*, Cincinnati, Ohio, May 14-17, 1990, pp. 625-631.

[ChB90b] Chirikjian, G.S., Burdick, J.W., "Kinematics of Hyper-Redundant Manipulators," *Proc. ASME Mechanisms Conference*, Chicago, IL, DE-Vol. 25, pp. 391-396, Sept. 16-19, 1990.

[ChB90c] G.S. Chirikjian and J.W. Burdick, "Applications of Hyper-Redundant Manipulators for Space Robotics and Automation," *Proc. Int. Symposium on Artificial Intelligence*

and Robotics Applications to Space, Kobe, Japan, November, 1990.

[ChB91a] Chirikjian, G.S., Burdick, J.W., "Parallel Formulation of the Inverse Kinematics of Modular Hyper-Redundant Manipulators," *Proc. IEEE Int. Conf. on Robotics and Automation*, Sacramento, California, April, 1991, pp. 708-713.

[ChB91b] Chirikjian, G.S., Burdick, J.W., "Kinematics of Hyper-Redundant Locomotion with Applications to Grasping," *Proc. IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA, April, 1991.

[ChB92a] Chirikjian, G.S., Burdick, J.W., "On the Determination of Kinematically Optimal Hyper-Redundant Manipulator Configurations," *Proc. IEEE Int. Conf. on Robotics and Automation*, Nice, France, May 10-15, 1992.

[ChB92b] Chirikjian, G.S., Burdick, J.W., "Design, Implementation, and Experiments with a 30 Degree-of-Freedon Hyper-Redundant Robot," *Proc. Int. Symp. Robotics and Manufacturing*, Albuquerque, NM, Nov., 1992.

[ChB92c] Chirikjian, G.S., Burdick, J.W., "A Modal Approach to Hyper-Redundant Manipulator Kinematics," submitted to *IEEE Trans. on Robotics and Automation.*

[ChB93a] Chirikjian, G.S., Burdick, J.W., "The Kinematics of Planar Hyper-Redundant Robotic Locomotion," submitted to *IEEE Trans. on Automation and Robotics.*

[ChB93b] Chirikjian, G.S., Burdick, J.W., "Curvilinear Hyper-Redundant Robotic Locomotion Over Uneven Terrain, with Applications to Grasping," submitted to *IEEE Trans. on Automation and Robotics.*

[ChB93c] Burdick, J.W., Radford, J., Chirikjian, G.S., "A 'Sidewinding' Locomotion Gait for Hyper-Redundant Robots," submitted to *IEEE Int. Conf. on Robotics and Automation*, Atlanta, GA, May, 1993.

[Ci] Ciarcia, S. "An Ultrasonic Ranging System." *Byte Magazine*, October, 1984, pp.113-123.

[HiU] Hirose, S., Umetani, Y., "Kinematic Control of Active Cord Mechanism With Tactile Sensors," *Proceedings of Second International CISM-IFT Symposium on Theory and Practice of Robots and Manipulators*, pp. 241-252, 1976.

[Gr] Greville, T.N.E. "The Pseudoinverse of a Rectangular or Singular Matrix and its Applications to the Solutions of Systems of Linear Equations," SIAM Review 1(1),1959,pp.38-43.

[Lat] J.C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, Boston, 1991.

[Li] Liegeois, A. "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms," IEEE Trans. Systems, Man and Cybernetics, 1977, Vol. SMC-7, No.12, pp.868-871.

[LoWes] Lozano-Perez, T. and Wesley, M.A. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications ACM*, Oct. 1979, Vol ACM 22, pp.560-570.

[MaKl] Maciejewski, A. and Klein, C.A. "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments," International Journal of Robotics Research, Vol.4, No.3, Fall, 1985, pp.109-117.

[Nak] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task Priority Based Redundancy Control of Robot Manipulators," *International Journal of Robotics Research*, vol. 6, 1987, pp. 3-15.

[Pe] Penrose, R. "On Best Approximate Solutions of Linear Matrix Equations," Proc. Cambridge Phill.,1956, Soc.52,pp.17-19.

[ReLu] Reznik, D. Lumelsky, V. "Motion Planning with Uncertainty for Highly Redundant Kinematic Structures I. 'Free Snake' Motion," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh, N.C., 1992.

[Sh] Sharir, "Algorithmic Motion Planning for Robots."

[TCB] Takanashi, N., Choset, H., and Burdick, J.W. "Local Sensor Based Planning for Hyper-redundant Manipulator," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Yokohama, Japan, 1993.

[Yo] Yoshikawa, T. "Analysis and control of robot manipulators with redundancy," Robotics research: The First International Symposium, ed. Brady, M. and Paul, R. Cambridge:MIT Press,1984, pp.735-747.

AIAA-94-1295-CP

FAULT TOLERANT KINEMATIC CONTROL OF HYPER-REDUNDANT MANIPULATORS

Nazareth S. Bedrossian[1]
The Charles Stark Draper Laboratory, Inc.
2200 Space Park Drive, Suite 210
Houston, TX 77058
e-mail: naz@mickey-csdl.jsc.nasa.gov

Abstract

This paper investigates the fault-tolerant control of hyper-redundant spatial manipulators. The standard resolved rate control law using the pseudoinverse is modified to account for joint failures. To combat the problem of extremely high joint velocity solutions generated near singular configurations by the pseudoinverse, the singularity robust inverse is employed. A method to compute an optimal scale factor for the robust inverse is derived. Simulation results of this approach applied to an 11 DOF manipulator are presented which verify the validity of this approach.

1 Introduction

Recent advances in the field of robotics has resulted in redundant manipulators gaining increased attention due to advantages over conventional manipulators such as increased dexterity. Hyper-redundant manipulators represent the next step in manipulator evolution. These manipulators posses a large number of Degrees-Of-Freedom (DOF). This paper investigates the kinematic control of hyper-redundant spatial manipulators with particular emphasis on fault-tolerant control.

The redundant structure of such manipulators endows them with many desirable properties, such as fault-tolerant features. The redundancy can be exploited to seamlessly complete end-effector tasks in the face of single or multiple joint motor failures. A rate-inverse kinematic control algorithm utilizing the pseudoinverse is presented that is insensitive to arbitrary joint motor failures. Another feature of redundant manipulators is the possibility of optimal joint-motion coordination. However, redundant manipulators are plagued by the presence of singular configurations. In these configurations first order motion in a certain end-effector direction is not possible, and are associated with loss of manipulator Jacobian matrix rank. Standard kinematic control algorithms fail at or near singular configurations. The singularity problem is also considered by implementing a singularity robust kinematic control algorithm that is insensitive to joint failures. The robustness is determined by a scaling factor. An optimal method to pick the scaling factor is also derived such that end-effector tracking accuracy is sacrificed in order to not violate joint velocity limits. Simulation results of this approach applied to an 11 DOF spatial manipulators are presented which verify the validity of the proposed methodology.

[1]Member Technical Staff, Control & Decision Systems.

[1]Copyright ©1993 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved

2 Manipulator Kinematics

In this section, a brief review of manipulator spatial kinematics is presented. For an n-link, serial, open-loop spatial manipulator, denote the space of joint coordinates by $q \in \mathcal{Q} = T^n$. The corresponding end-effector position and attitude is denoted by $x \in \mathcal{X} = \mathcal{R}^3 \times SO(3)$. For a redundant manipulator, $n > 6$. The end-effector motion kinematics are given by:

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} = J(q)\,\dot{q}$$

Here, p denotes the end-effector position, ω is the end-effector angular velocity expressed in an inertial Cartesian reference frame, and $J(q)$ is the $6 \times n$ "constructed" Jacobian transformation matrix.

All manipulators posses singular configurations inside their workspace. These configurations, q^s, are manifest when the Jacobian matrix loses full rank, i.e. $rank(J) < dim(\mathcal{X})$. In terms of the Singular Value Decomposition (SVD) of the Jacobian matrix, $J = U\Sigma V^T$, this corresponds to at least one singular value $\sigma_i = 0$. The corresponding singular direction U_i

(the i-th column of U) is the direction in which end-effector motion to first order is instantaneously impossible. At a singular configuration, a solution for the joint rates cannot be constructed from the first order approximation of the forward kinematics. Hence, kinematic control algorithms that rely on inversion of the Jacobian matrix in one form or the other generate extremely large joint rate solutions near or at a singular configuration.

3 Inverse Rate Kinematics

Manipulator tasks are described in end-effector space, $\mathcal{X}$, while actuator commands are in joint-space, $\mathcal{Q}$. For this reason, an inverse kinematic algorithm is required to generate joint-angle commands from the end-effector commands. The inversion can be carried out at different differential levels of the forward kinematics. In this paper only first order inversion or Inverse Rate Kinematics (IRK) will be considered. Kinematic control algorithms that use this approach are also known as Resolved Rate Control (RRC) algorithms.

For inverse rate kinematics, the end-effector velocity profile along the desired trajectory is specified. At each increment, the joint rates are computed by "inverting" the Jacobian matrix evaluated at the current configuration. These joint rates are then integrated (usually via Euler integration) to generate the next set of joint angles. Assuming the current configuration is nonsingular, the general form of the solution for joint rates is,

$$\dot{q}_k \;=\; J^{\dagger}(q_k)\,\dot{x}_k \;+\; \nu$$

where the notation $J^{\dagger}$ denotes a generalized inverse and ν is the instantaneous self motion (or null motion), i.e. $J(q_k)\,\nu = 0_{m\times 1}$. For nonredundant manipulators $J^{\dagger} = J^{-1}$ and $\nu = 0_{n\times 1}$.

For a redundant manipulator $J^{\dagger} = J^T\left[J\,J^T\right]^{-1}$, known as the pseudoinverse of J, and $\nu \neq 0_{n\times 1}$. The pseudoinverse originates from a 2-norm joint rate minimization problem. Specifically it is obtained as the solution to the following quadratic cost optimization problem:

$$\begin{aligned} \min_{\dot{q}} \quad & 0.5\,\dot{q}^T\dot{q} \qquad (1)\\ \textit{subject to} \quad & J(q)\dot{q} = \dot{x} \end{aligned}$$

In the redundant case, there are $n-6$ degrees of freedom in the nullspace of the Jacobian matrix that can be assigned arbitrarily. In both cases, the joint angles are then obtained from:

$$q_{k+1} \;=\; q_k \;+\; \dot{q}_k\,\Delta t$$

4 Resolved Rate Law For Failed Joints

The solution to the quadratic optimization problem in the previous section assumed that all the joints were active or had not failed. In this section a modified version of the pseudoinverse which accounts for joint failures is derived. Assuming joint i fails, this implies that $\dot{q}_i = 0$. The incidence of a failed joint can be expressed as an additional constraint of the form:

$$A\dot{q} = 0_{m\times 1}$$

The $m \times n$ constraint matrix, A, is a zero matrix with $A(i,j) = 1$ for the j-th failed joint with a total number of failures equal to m. To illustrate this, for a 9-link manipulator with joints 4 and 7 failed, the corresponding A matrix has the form:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

To derive the modified pseudoinverse to account for failed joints, the quadratic optimization problem (2) is reformulated as:

$$\min_{\dot{q}} \quad 0.5\,\dot{q}^T\dot{q} \qquad (2)$$

$$\begin{aligned} \textit{subject to} \quad & J(q)\dot{q} \;=\; \dot{x} \\ \textit{subject to} \quad & A\dot{q} \;=\; 0_{m\times 1} \end{aligned}$$

The solution of this new optimization problem (2) is,

$$\dot{q} = B\,J^T\left[\,J\,B\,J^T\,\right]^{-1}\dot{x} \qquad (3)$$

where:

$$B = \left[\,1_{n\times n} - A^T\left(A\,A^T\right)^{-1}A\,\right]$$

Comparing this solution with the one for all joints free, it is seen that by setting $A = 0_{m\times n}$ the pseudoinverse solution is obtained.

5 Robust RRC For Failed Joints

The standard IRK solution utilizing the pseudoinverse fails at or near singular configurations due to extremely large joint rate solutions. It has been shown [2] that the pseudoinverse does not generate singularity free trajectories. Therefore, the pseudoinverse approach does not posses any singularity avoidance properties. Since solving (2) can drive the system to a singular configuration, one approach to alleviate this problem is to relax the equality trajectory constraint.

The Singularity Robust Inverse (SRI) (or damped least squares) proposed in [6], and [8] as an alternative to the pseudoinverse accomplishes a tradeoff between

accuracy and feasibility of solution. Near a singular configuration the joint rates remain finite and bounded in exchange for a build-up in tracking error. The robust resolved rate law is obtained from the solution to the constrained minimization problem,

$$\min_{\dot{q}} \quad 0.5\, e^T W\, e \tag{4}$$

where:

$$e = \begin{bmatrix} \dot{x} - J(q)\dot{q} \\ \dot{q} \end{bmatrix}$$

$$W = \begin{bmatrix} W_1 & 0_{6\times n} \\ 0_{n\times 6} & W_2 \end{bmatrix}$$

The weighting matrices, W_1 (6×6) and W_2 $(n \times n)$, are arbitrary. Assuming $W_1 = 1_{6\times 6}$ and $W_2 = \kappa\ 1_{n\times n}$, the solution to (4) is given by [6]:

$$\dot{q} = J^*\dot{x} = \left[\, J^T J \,+\, \kappa I \,\right]^{-1} J^T \dot{x} \tag{5}$$

In the above, J^* is the Singularity Robust Inverse (SRI). An alternative but equivalent expression for J^* is [6]:

$$\dot{q} = J^*\dot{x} = J^T \left[\, J\, J^T + \kappa I \,\right]^{-1} \dot{x} \tag{6}$$

Setting the scaling factor $\kappa = 0$ in (11), it is seen that J^* reduces to the standard pseudoinverse $J^\dagger$. The error vector e represents the tradeoff between accuracy of solution (expressed by $e(1)$) and feasibility of solution (expressed by $e(2)$). The tradeoff parameter is the scaling factor, κ, which is the degree of freedom in the formulation of the SRI control law. In the following section, an optimal method to chose the scaling factor is presented.

The standard formulation of the SRI assumes that all joints are free. To derive the modified SRI to account for failed joints, the optimization problem (4) is reformulated as:

$$\min_{\dot{q}} \quad 0.5\, e^T W\, e \tag{7}$$

$$subject\ to \quad A\dot{q} \,=\, 0_{m\times 1}$$

The solution to this new optimization problem (7) is given by,

$$\dot{q} = P^{-1}\left[\, I - A^T \left[A\, P^{-1} A^T\,\right]^{-1} A\, P^{-1}\,\right] J^T W_1\, \dot{x} \tag{8}$$

where:

$$P = J^T W_1\, J \,+\, W_2$$

It will be assumed that $W_1 = 1_{6\times 6}$ and $W_2 = \kappa\ 1_{n\times n}$. It is noted that the first term in the right hand side of (8) is just the standard SRI solution, while the second term accounts for the effect of the failed joints.

5.1 Determining the Scaling Factor

The robustness properties or the SR-Inverse are determined by the scaling factor κ, which expresses the tradeoff between the exactness and feasibility of solution. Depending on the particular emphasis of an application, several methods for choosing the scaling factor have appeared in the literature [6], [8], [4]. These methods adjust the scaling factor as a function of some Jacobian based metric such as the minimum singular value or the manipulability measure. Hence, the scaling factor is adjusted based on the proximity of the manipulator to a singular configuration.

Another approach is to chose the scaling factor such that an appropriate norm of the joint rates does not exceed a predetermined threshold [3], [5]. Such an approach directly takes into account the maximum allowable joint velocity constraints while minimizing the end-effector deviation from the desired trajectory. Whenever the pseudoinverse (or modified pseudoinverse in the case of failed joints) does not violate the joint velocity threshold (i.e. the solution is feasible) scaling is not required and $\kappa = 0$. In this case the SRI solution is not required. When the pseudoinverse solution violates the velocity norm threshold (i.e. the solution is infeasible) the SRI can be used to generate a feasible solution which minimizes the deviation from the specified end-effector trajectory if the choice of scaling factor satisfies:

$$\| \dot{q} \|_m \;=\; \| J^T \left[\, J\, J^T \,+\, \kappa\, I \,\right]^{-1} \dot{x} \|_m \;\leq\; \dot{q}_{max} \tag{9}$$

In (9), the notation $\| \cdot \|_m$ denotes the vector m-norm. Hence, the problem of generating joint rates that do not violate a rate limit is posed as follows:

$$\begin{aligned} &if\ \| J^\dagger \dot{x} \|_m \;\leq\; \dot{q}_{max} \quad \kappa = 0 \quad else\ \ solve \\ &\| J^T \left[\, J\, J^T \,+\, \kappa\, I \,\right]^{-1} \dot{x} \|_m \;=\; \dot{q}_{max} \end{aligned} \tag{10}$$

Obtaining a closed form solution to (10) is not practically possible as it is a nonlinear problem. An iterative approach to solve (10) for $m = 2$ has been presented in [3] and [5]. However, an approximate closed form solution can be obtained which is described in the following.

Using the singular value decomposition (SVD) of the Jacobian matrix, $J = U\Sigma V^T$, (10) can be written in the form,

$$\| \dot{q} \|_m \;=\; \| V\, \Sigma^\dagger\, U^T\, \dot{x} \|_m \;=\; \dot{q}_{max} \tag{11}$$

where:

$$\Sigma^{\dagger} = \begin{bmatrix} \frac{\sigma_1}{\sigma_1^2+\kappa} & 0 & \cdots & 0 \\ 0 & \frac{\sigma_2}{\sigma_2^2+\kappa} & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & \frac{\sigma_6}{\sigma_6^2+\kappa} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

To simplify (11), the joint velocity norm can be upper-bounded using induced matrix norms [7]:

$$\| \dot{q} \|_m \leq \| V \|_{im} \| \Sigma^{\dagger} \|_{im} \| U^T \dot{x} \|_m \qquad (12)$$

In (12), the notation $\| \cdot \|_{im}$ denotes the induced (m) norm corresponding to the vector norm $\| \cdot \|_m$. Using (12), a conservative relation involving the joint velocity norm threshold is:

$$\| V \|_{im} \| \Sigma^{\dagger} \|_{im} \| U^T \dot{x} \|_m \leq \dot{q}_{max} \qquad (13)$$

Solving for $\Sigma^{\dagger}$ from (13):

$$\| \Sigma^{\dagger} \|_{im} \leq \frac{\dot{q}_{max}}{\| V \|_{im} \| U^T \dot{x} \|_m} \qquad (14)$$

The relation (14) is the key to solving for the scaling factor in order to enforce the joint velocity norm limits. Typically, joint velocity thresholds are specified as either a 2-norm or ∞-norm constraint. For a 2-norm rate limit, the induced 2-norm of $\Sigma^{\dagger}$ is [7],

$$\| \Sigma^{\dagger} \|_{i2} = \left[\lambda_{max} \left({\Sigma^{\dagger}}^T \Sigma^{\dagger} \right) \right]^{1/2}$$

where:

$$\lambda_{max} \left({\Sigma^{\dagger}}^T \Sigma^{\dagger} \right) = \text{Maximum eigenvalue of } {\Sigma^{\dagger}}^T \Sigma^{\dagger}$$

For an ∞-norm rate limit, the induced ∞-norm of $\Sigma^{\dagger}$ is [7]:

$$\| \Sigma^{\dagger} \|_{i\infty} = \max_i \sum_j |\Sigma^{\dagger}_{i,j}|$$

Due to the special structure of $\Sigma^{\dagger}$ and the fact that $\Sigma^{\dagger}_{i,j} > 0$, it is easy to show that in both cases its induced norm reduces to:

$$\| \Sigma^{\dagger} \|_{i2} = \| \Sigma^{\dagger} \|_{i\infty} = \max_i \frac{\sigma_i}{\sigma_i^2+\kappa}$$

In applications, usually the maximum absolute joint velocity is limited instead of the quadratic velocity norm. Assuming that maximum absolute velocity limit is the same for all joints, (14) reduces to:

$$\max_i \frac{\sigma_i}{\sigma_i^2+\kappa} \leq \frac{\dot{q}_{max}}{\| V \|_{i\infty} \| U^T \dot{x} \|_{\infty}} \qquad (15)$$

For a fixed value of the scaling factor κ, as a singular value σ_i approaches zero, the expression $\sigma_i/(\sigma_i^2+\kappa)$ increases until it reaches a maximum value when $\sigma_i = \sqrt{\kappa}$ and then decreases to zero. Therefore,

$$\max_i \frac{\sigma_i}{\sigma_i^2+\kappa} \leq \frac{1}{2\sqrt{\kappa}}$$

Hence, a conservative solution for the scaling factor that will satisfy (15) is:

$$\kappa \geq \left[\frac{\| V \|_{i\infty} \| U^T \dot{x} \|_{\infty}}{2\, \dot{q}_{max}} \right]^2 \qquad (16)$$

In actual implementation, the equality is used in computing the scaling factor. Similarly, if the joint velocity threshold is expressed in terms of a 2-norm, the solution for the scaling factor is given by:

$$\kappa \geq \left[\frac{\| V \|_{i2} \| U^T \dot{x} \|_2}{2\, \dot{q}_{max}} \right]^2$$

6 Numerical Simulations

In this section, simulation results of the techniques described in this paper applied to an 11 DOF spatial manipulator are presented. This is a special modular manipulator developed at NASA/JSC [1] with a 15 foot reach. Its architecture is described by the joint sequence RPR-PR-PR-PR-RPR where R denotes a roll joint and P denotes a pitch joint.

The first simulation example is used to illustrate the fault-tolerant resolved rate law using the modified pseudoinverse (3). The initial configuration of the manipulator is given by $q_0 = [\,45,\ -60,\ 0,\ -45,\ 0,\ 45,\ 90,\ 45,\ 0,\ 0,\ 0,\ 0\,]$. The initial configuration is shown in Figure 1. The end-effector command is $\dot{x} = [\,-5.5,\ -2.5,\ 1.5,\ 0,\ 0,\ 0\,]$ with units of in/sec for the translational component and deg/sec for the rotational component. In the following figures, the end-effector position is displayed as, solid line for x-axis, dashed line for y-axis, and dotted line for z-axis. The end-effector attitude is given as a Pitch, Yaw, Roll (PYR) Euler angle sequence, with solid line for pitch, dashed line for yaw, dotted line for roll. For all joints free throughout the simulation, the results for the standard pseudoinverse are shown in Figures 2 and 3. Figure 2 shows the end-effector position and attitude while

Figure 3 shows the joint angle trajectory. Now consider the case when joints 1,6, and 7 have failed throughout the simulation. The results for the modified pseudoinverse (3) are shown in Figures 4 and 5. Figure 4 shows the end-effector trajectory which is identical to the all joints free case. Figure 5 shows the joint angle trajectory. It is seen that joints 1,6, and 7 do not move as would be expected.

To illustrate the SRI with joint velocity thresholds, consider a maneuver of moving the end-effector in the x-direction. The initial configuration is $q_0 = [\,0,\ -50,\ 0,\ -30,\ 0,\ 70,\ 0,\ 30,\ 0,\ 0,\ 0\,]$. The end-effector command is $\dot{x} = [\,-5,\ 0,\ 0,\ 0,\ 0,\ 0\,]$. First consider the case when all joints are free throughout the simulation. The results of using the standard pseudoinverse are shown in Figures 6 and 7. Figure 6 shows the position and attitude history of the end-effector, while Figure 7 shows the joint rate profile. It is seen that the maximum joint rate is less than 6 deg/sec.

Now consider the case when joints 2,4, and 7 have failed throughout the simulation. The results of using the modified pseudoinverse, (3), are shown in Figures 8 and 9. The end-effector trajectory shown in Figure 8 is seen to deviate somewhat from the desired trajectory. This is due to the very high joint rate solution (up to 400 deg/sec) generated by (3) shown in Figure 9, and the particular integration approach used to generate the end-effector trajectory. The results of using the SRI with a maximum absolute joint rate limit of 50 deg/sec, i.e. $\| \dot{q} \|_{i\infty} \leq 50\ deg/sec$, are shown in Figures 10 to 12. The end-effector trajectory is shown in Figure 10 where the deviation from the desired trajectory appears in the x and z-axes and pitch angle. The main effect of using the SRI is the reduction in system response in the commanded direction. From Figure 10 it is seen that at the end of the simulation the manipulator has only moved halfway in the x-direction. The joint angle trajectory is shown in Figure 11 from which it is seen that the joint rate limit is not exceeded. Finally, Figure 12 shows the history of the SRI scale factor. It is seen that the SRI solution is activated at approximately 2 seconds.

7 Conclusion

This paper has addressed the fault tolerant kinematic control of hyper-redundant manipulators. The standard resolved rate control law utilizing the pseudoinverse was modified to account for joint failures. However, pseudoinverse based control laws fail at or near singular configurations due to extremely high joint rate solutions. To generate feasible joint motions near singular configurations, the Singularity Robust Inverse (SRI) instead of the pseudoinverse was employed. The standard formulation of the SRI was modified so as to account for the possibility of failed joints. As the robustness properties of the SRI are determined by the choice of scaling factor, an optimal method to pick the scaling factor was presented. Numerical simulations were presented utilizing an 11 DOF spatial manipulator to illustrate the proposed solution methodologies. The simulation results verified the validity of the proposed methods.

References

[1] Ambrose, R., "Space Modular Manipulators", NASA/JSC presentation SMM 93-13, March 1993.

[2] Baillieul, J., Hollerbach, J., and Brockett, R., "Programming and Control of Kinematically Redundant Manipulators", *23rd IEEE Conf. Dec. & Cont.*, 1984, pp. 768-774.

[3] Deo, A. S., and Walker, I. D., "Robot Subtask Performance with Singularity Robustness using Optimal Damped Least-Squares", *1992 IEEE Int. Conf. Rob. & Aut.*, pp. 434-441.

[4] Egeland, O., Sagli, J. R., Spangelo, I., and Chiaverini, S., "A Damped Least-Squares Solution to Redundancy Rresolution", *1991 IEEE Int. Conf. Rob. & Aut.*, pp. 945-950.

[5] Maciejewski, A. A., and Klein, C. A., "The Singular Value Decomposition : Computation and Application to Robotics", *Int. J. Rob. Res.*, Vol. 8, No. 6, 1989, pp. 63-79.

[6] Nakamura, Y., and Hanafusa, H., "Inverse Kinematic Solutions With Singularity Robustness for Robot Manipulator Control", *ASME Journal Dynamic Systems, Measurement and Control*, Vol. 108, Sept. 1986, pp. 163-171.

[7] Vidyasagar, M., *Nonlinear Systems Analysis*, Prentice-Hall, Inc., 1978.

[8] Wampler, C. W., "Manipulator Inverse Kinematic Solutions Based on Damped Least-Squares Solutions", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-16, No. 1, January/February 1986, pp. 93-101.

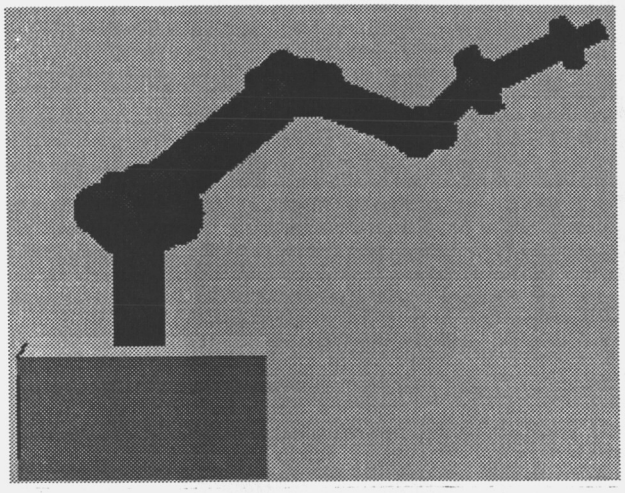


Figure 1: Modular manipulator in initial configuration

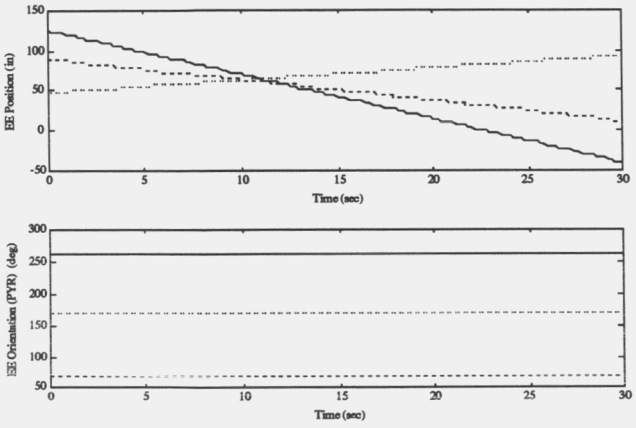


Figure 2: Pseudoinverse solution for all joints free: End-effector position and attitude

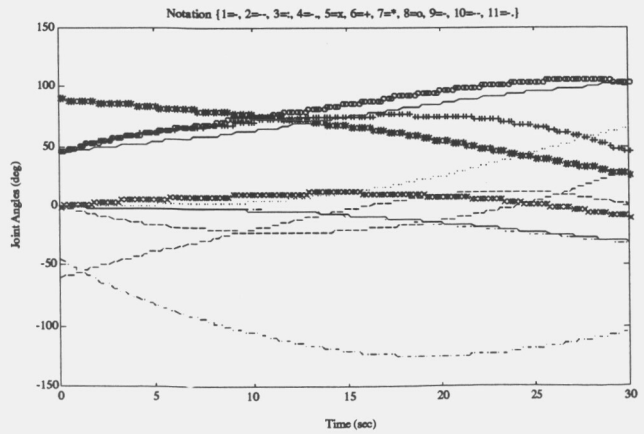


Figure 3: Pseudoinverse solution for all joints free: Joint angles

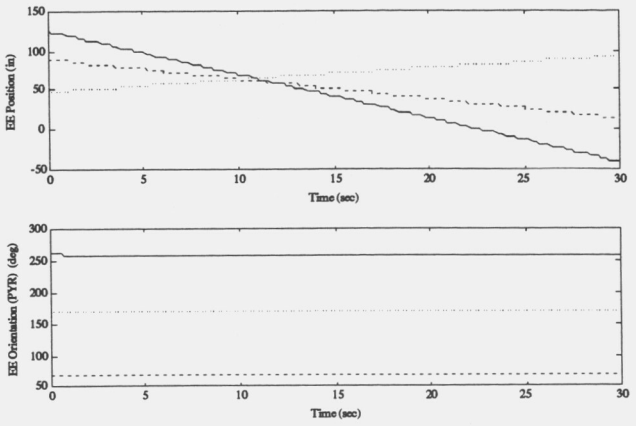


Figure 4: Pseudoinverse solution for all joints free: End-effector position and attitude

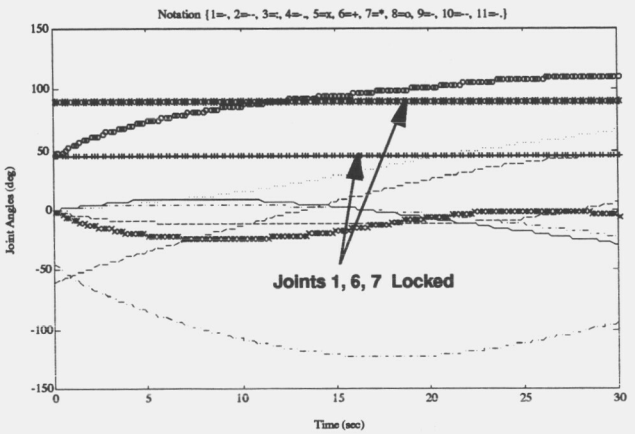


Figure 5: Pseudoinverse solution for all joints free: Joint angles

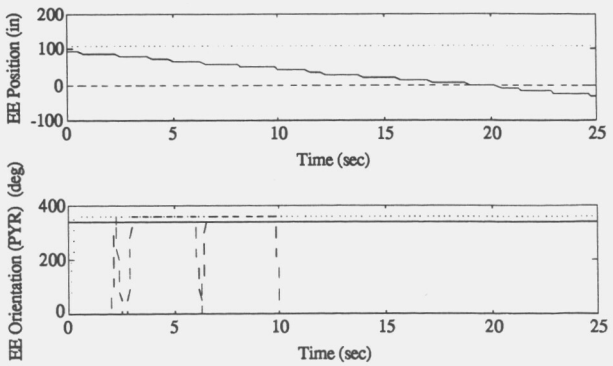


Figure 6: Pseudoinverse solution for all joints free: End-effector position and attitude

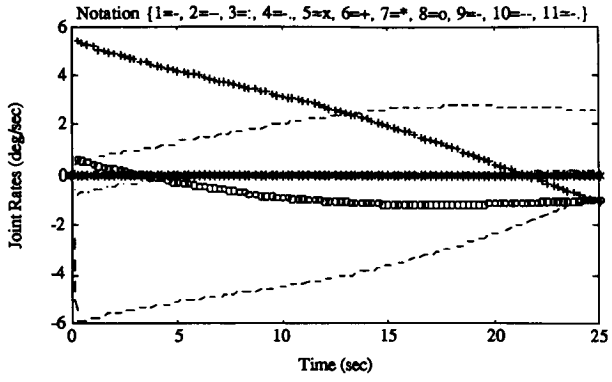


Figure 7: Pseudoinverse solution for all joints free: Joint rates

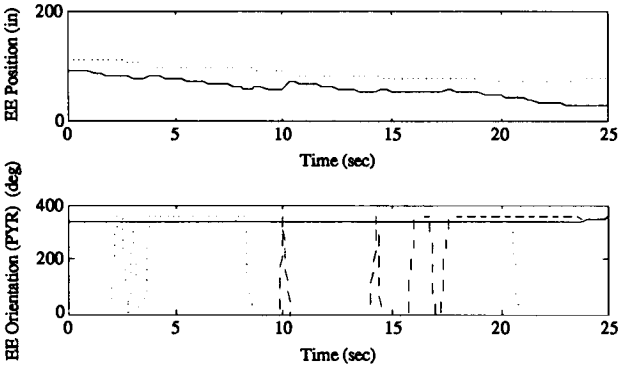


Figure 8: Pseudoinverse solution for 3 failed joints: End-effector position and attitude

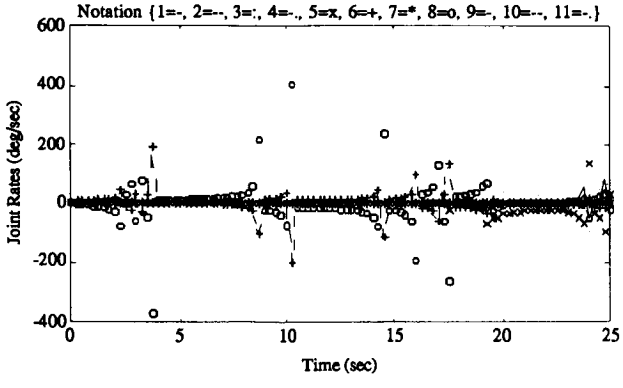


Figure 9: Pseudoinverse solution for 3 failed joints: Joint rates

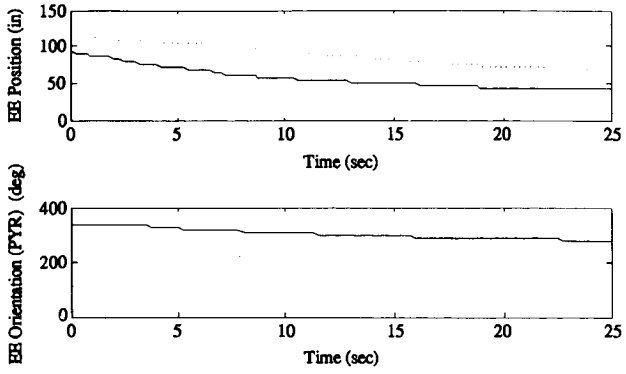


Figure 10: SR-Inverse solution for 3 failed joints: End-effector position and attitude

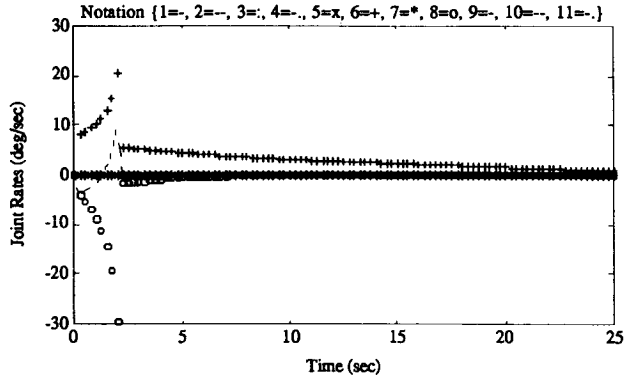


Figure 11: SR-Inverse solution for 3 failed joints: Joint rates

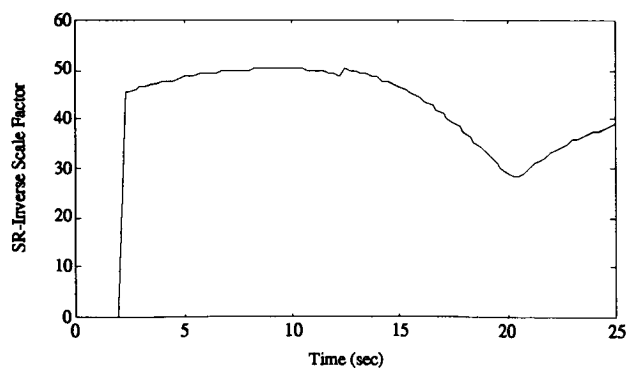


Figure 12: SR-Inverse solution for 3 failed joints: Scale Factor

AIAA-94-1296-CP

Failure Tolerant Operation of Kinematically Redundant Manipulators

Christopher L. Lewis Anthony A. Maciejewski
School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Abstract— **The high cost involved in the retrieval and repair of robotic manipulators used for remediating nuclear waste, processing hazardous chemicals, or for exploring space or the deep sea, places a premium on the reliability of the system as a whole. For such applications, kinematically redundant manipulators are inherently more reliable since the additional degrees of freedom (DOF) may compensate for a failed joint. In this work, a redundant manipulator is considered to be fault tolerant with respect to a given task if it is guaranteed to be capable of performing the task after any one of its joints has failed and is locked in place. A method is developed for insuring the failure tolerance of kinematically redundant manipulators with respect to a given critical task. Techniques are developed for analyzing the manipulator's workspace to find regions which are inherently suitable for critical tasks due to their relatively high level of failure tolerance. Then, constraints are imposed on the range of motion of the manipulator to guarantee that a given task is completable regardless of which joint fails.**

I. Introduction

Kinematically redundant manipulators have been proposed for use in the cleanup and remediation of nuclear and hazardous materials, as well as for remote applications such as deep space or sea exploration, where repair of broken actuators and sensors is impossible and the probability of their failure is increased due to the harsh operating environment [2], [3]. In these situations the extra degrees of freedom of a redundant manipulator may be used to compensate for the failed joints if the manipulator has been properly designed and controlled. The most basic task of a manipulator, i.e. the positioning/orienting the end effector in the workspace, is described by the forward kinematic equation

$$x = f(\theta), \tag{1}$$

where $x \in R^m$ is the generalized vector of the position/orientation of the end effector and $\theta \in R^n$ is the vector of joint variables. In this framework, point to point tasks can be described by a series of end-effector positions to be obtained at desired times, i.e., $x(t_i)$, with a kinematic inverse equation

$$\theta = f^{-1}(x) \tag{2}$$

being solved to determine the corresponding required joint values, $\theta(t_i)$. A kinematically redundant manipulator can, in general, satisfy an end effector positioning constraint, $x(t_i)$, with an infinite family of joint values satisfying (2). The underlying premise for advocating the use of redundant manipulators for critical applications is that if a joint should fail, then the redundancy of the manipulator may permit the completion of the task. Although commercial manipulators currently are not equipped with the necessary circuitry to detect failures and apply the brakes to any failing joint, the need for such a mechanism is well known [12],[13]. If failed joints are locked, then, a single joint failure reduces the number of degrees of freedom (DOF) of the system by one, and the new kinematic functions $\hat{f}()$ and their inverses $\hat{f}^{-1}$ differ markedly from the original ones.

In [12] a method is described for designing manipulators to be fault tolerant with regards to a given point to point task. They assume that any joint may fail anywhere within its entire range of motion. A manipulator is said to be fault tolerant with respect to a given set of task points $x(t_i)$ only if there exist solutions to (2) for every possible failure. With this assumption, the worst case typically occurs when a failing joint is folded in on itself. In the work described here, failure tolerance is achieved by imposing constraints on the motion of all joints prior to a failure. By judiciously selecting the specific solution from the family of solutions to (2), the worst case need not occur. Thus failure tolerance may be achieved with less complex manipulator designs, and for manipulators not originally designed with failure tolerance in mind.

An alternative to defining the manipulator's task as a sequence of end-effector positions is to specify the end-effector velocity profile. At the velocity level, the kinematic equations relating the joint rates $\dot{\theta}$ to the end-effector's velocity $\dot{x}$ are given by

$$\dot{x} = J\dot{\theta} \tag{3}$$

where $J \in R^{m \times n}$ is the manipulator Jacobian matrix which is a function of the manipulator's configuration.

This work was supported by Sandia National Laboratories under contract number 18-4379B. Additional funding was provided by the National Science Foundation under grant CDR 8803017 to the Engineering Research Center for Intelligent Manufacturing Systems.

opyright © 1993 American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

The solution for all joint rates that satisfy the desired end-effector velocity can be represented by

$$\dot{\theta} = J^{+}\dot{x} + (I - J^{+}J)z \tag{4}$$

where $^{+}$ indicates the pseudoinverse, $(I - J^{+}J)$ is the projection onto the null space, and z represents an arbitrary vector in the joint velocity space [8]. The second term in this equation clearly indicates that there is a family of joint trajectories that satisfy (3). However, unlike the kinematic function $f()$ relating the joint values to the end-effector's position, the Jacobian for the failed system is easily derived from the original system's Jacobian by zeroing the column of the failed joint. Using this fact it is possible to develop an inverse kinematic function which insures that the manipulator will have some degree of local dexterity after an arbitrary joint failure [7]. The measure of dexterity in this case is defined as the smallest singular value of the Jacobian, σ_m, so that a kinematic failure tolerance measure, kfm, is given by

$$kfm(\theta) = \min_{f=1-n} \sigma_m({}^{f}J) \tag{5}$$

where ${}^{f}J$ is the manipulator Jacobian matrix for the system with its f'th joint locked. Having a large value for $kfm(\theta)$ insures that after an arbitrary joint failure the manipulator will still be able to satisfy an arbitrary desired end-effector velocity in the vicinity of the failure. Unfortunately, this measure is inherently local in nature and can not guarantee that the complete trajectory remains feasible after the failure. However, it will be shown that the local failure tolerance measure, $kfm(\theta)$, can be used to guide the search for regions within the workspace for which one can insure that the entire desired task can be completed regardless of joint failures.

The remainder of this paper is organized as follows. First, a method for analyzing the fault tolerance of a given location in the workspace is discussed. Second, the constraints necessary to guarantee fault tolerance for a single point are described. Third, a procedure that uses the local measure of fault tolerance to identify candidate regions of the workspace where critical task should be placed is discussed. Then, a method for determining the constraints necessary to guarantee the fault tolerance of the manipulator with respect to the given critical path is outlined.

II. Surfaces of Self-Motion

For a kinematically redundant manipulator the family of joint configurations satisfying (1) forms an $(n-m)$-dimensional hyper-surface in the n-dimensional configuration space of the manipulator [1],[6]. Joint motion constrained to this hyper-surface does not affect the position/orientation of the end effector so that these hyper-surfaces are frequently referred to as self-motion manifolds. The null space of the manipulator's Jacobian given by the set of vectors satisfying (3) with $\dot{x} = 0$ defines the tangent plane to the self-motion manifold. As a simple example, consider the 3 DOF planar manipulator shown

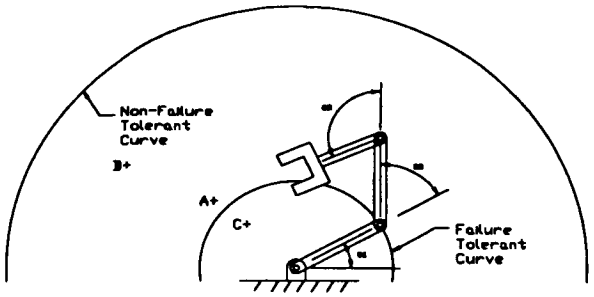


Fig. 1. A three degree of freedom planar manipulator with equal link lengths.

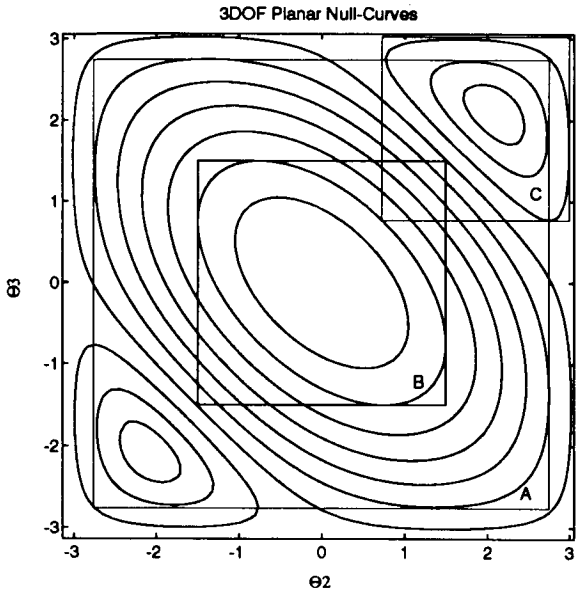


Fig. 2. The set of joint configurations having the manipulator's end-effector at a single location form curves in the configuration space of the manipulator. These curves are the self-motion surfaces for a 3 link planar manipulator. The self-motion surfaces for some regions of the workspace are markedly larger than others. Points with large self-motion surfaces tend to be more failure tolerant.

in Fig. 1 for which the self-motion manifolds are one-dimensional curves. For this manipulator a projection of the self-motion curves onto the $\theta_2 - \theta_3$ plane is shown in Fig. 2. Each curve represents the family of joint variable combinations which place the end-effector at a constant radius from the base. From the figure, it is clear that some regions of the workspace have larger self-motion manifolds than others. The two extremes occur at the boundary of the manipulator's workspace, which corresponds to the point at the origin of the configuration space, and on the circle which is centered at the base and has a radius of 1 meter. In the first case, the self-motion surface vanishes to a point. This fact indicates that the manipulator will not be capable of reaching the original boundary after any joint failure. At the other extreme, the self-motion curve spans the entire range of joint values, even in θ_1 which is not shown. This fact is significant since regardless of which joint fails, or where it fails, the manipulator will always be capable of tracing out the unit circle with it's end-effector. It is interesting to note, that the local failure

tolerance measure, (5), reaches its exact theoretically optimal value on the self-motion surface of this globally failure tolerant point. Also note that $kfm(0) = 0$ at the reach singularity. These attributes lead to the use of kfm as a first pass when evaluating the workspace in order to place critical task. Clearly, when a joint fails and is locked, the manipulator is more likely to be able to reach points with large self-motion surfaces than those with small ones.

To guarantee that a manipulator is able to return to a desired workspace location, one must, in general, constrain the motion range for each of the n joints. The minimum and maximum joint values of the ith joint, denoted $\theta_{i_{min}}$ and $\theta_{i_{max}}$, respectively, can be determined from the minimum and maximum values of θ_i over the entire self-motion manifold. This effectively superscribes an n-dimensional box aligned with the joint axes around the self-motion manifold. The size of this bounding box is an indication of the inherent failure tolerance of the workspace point for which it was computed. If the manipulator fails while operating within the bounding box of a given desired end-effector location x, then it will always be able to position its end-effector at that point regardless of where the end effector is located when the failure occurs. For example, consider again the 3DOF manipulator, for which the bounding boxes associated with the self-motion surfaces for the three workspace points labeled A, B, and C in Fig. 1 have been drawn in Fig. 2. Note, that although θ_1 and it's associated boundaries are not shown, they need to be considered. If the manipulator fails while within the boundary of any one of the bounding boxes, then the manipulator will always be able to position its end-effector at those points regardless of which joint fails. The region of the configuration space which lies inside all three bounding boxes is of particular interest. If the manipulator operates within this region, then regardless of which joint fails, it will be able to reach all three points. Unfortunately, it is not possible to reach points B and C and stay within this region of the joint space, however, it should be clear that obtaining the bounding region for self-motion manifolds reveals the potential failure tolerance of various locations within the workspace.

Several iterative methods exist in the literature for characterizing one dimensional self-motion curves [1],[4],[5],[9]. For systems with two or more degrees of redundancy an estimate of the size of the self-surface may be obtained by using a Jacobian iteration of the form

$$\dot{\theta} = \pm(I - J^+J)\hat{e}_i + J^+(x^* - x) \tag{6}$$

where $\hat{e}_i$ is a unit vector along the ith joint axis and x^* is the workspace end-effector location being evaluated. The first term represents motion along the self-motion manifold until the tangent to the manifold becomes orthogonal to the joint axis direction $\hat{e}_i$. The second term is required to compensate for any errors that are accumulated during the iterative procedure [5]. This method is effective for one-dimensional self-motion curves as in the 3DOF planar case, but may yield an insufficiently low estimate for self-motion manifolds of higher dimensions.

For a two-dimensional self-motion surface, a simple and effective method for estimating the bounds of the self-motion surface is to iteratively trace out a linearly increasing spiral on the self-motion surface. Keeping track of the values obtained by each joint along the spiral provides an estimate of the bounding box containing the self-motion surface. A non-escaping spiral, depicted in Fig. 3, has a parameterized equation of the form

$$\begin{aligned} \dot{\phi} &= \tfrac{v}{r} \\ r &= \gamma\phi \end{aligned} \tag{7}$$

where v is the velocity along the spiral, r and ϕ are the polar coordinates of the spiral, and γ controls the distance between successive rotations. Since this particular spiral passes within a controlled distance from every point in the plane, when it is transformed onto the self-motion surface it will tend to fill the surface. The iterative transformation procedure from parameter to configuration space is given by

$$\dot{\theta} = \sin(\phi)\hat{v}_{n-1} + \cos(\phi)\hat{v}_n + J^+(x^* - x) \tag{8}$$

where $\hat{v}_{n-1}$ and $\hat{v}_n$ are orthogonal unit vectors that span the null-space of the manipulator's Jacobian evaluated at the current configuration. The vectors $\hat{v}_n$ and $\hat{v}_{n-1}$ can be computed as the singular vectors from the singular value decomposition of J. Since $\hat{v}_{n-1}$ and $\hat{v}_n$ are not unique, one must be careful to ensure that vectors chosen are the ones nearest to those of the previous iteration. For example, if the current singular vectors are represented by $\hat{v}_{n-1}$ and $\hat{v}_n$ then once (8) is evaluated and used to update the manipulator configuration, the new Jacobian will in general have different singular vectors $\hat{v}'_{n-1}$ and $\hat{v}'_n$. To accurately reflect the continuous rotation of these two vectors as the null space rotates, one can use the following set of equations

$$\begin{aligned} \hat{v}'_{n-1} &= \lambda\hat{w}_1 + (1-\lambda)\hat{w}_2 \\ \hat{v}'_n &= (1-\lambda)\hat{w}_1 - \lambda\hat{w}_2 \end{aligned} \tag{9}$$

where

$$\lambda = \frac{(\hat{w}_1^T\hat{v}_{n-1})^2}{(\hat{w}_1^T\hat{v}_{n-1})^2 + (\hat{w}_2^T\hat{v}_{n-1})^2} \tag{10}$$

where $\hat{w}_1$ and $\hat{w}_2$ are any unit vectors that span the new null space. Note, that the sign should be examined to select the smallest resulting rotation. An ideal algorithm for computing the SVD that automatically calculates the continuous rotation of the null space is presented in [11]. An illustration of this technique for mapping out a two-dimensional self-motion surface is presented in Fig. 4. This figure shows a three-dimensional projection of the five-dimensional configuration space for a PUMA used in three-dimensional positioning tasks.

III. Joint Constraints to Guarantee Fault Tolerance

As was indicated in the previous section, a workspace location, x^*, may be guaranteed to be reachable regardless of joint failures if the manipulator is constrained to operate within the associated self-motion manifold's bounding

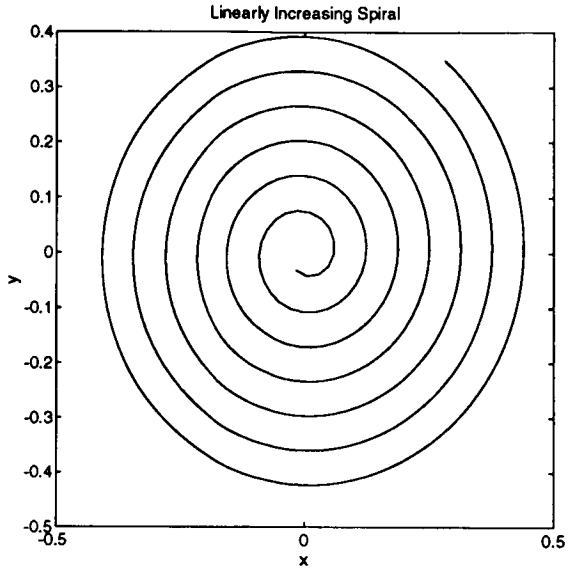


Fig. 3. A linearly increasing spiral passes within a controlled distance from every point in the plane, and thus it may be used to estimate the bounds of a 2D surface in an n-dimensional space.

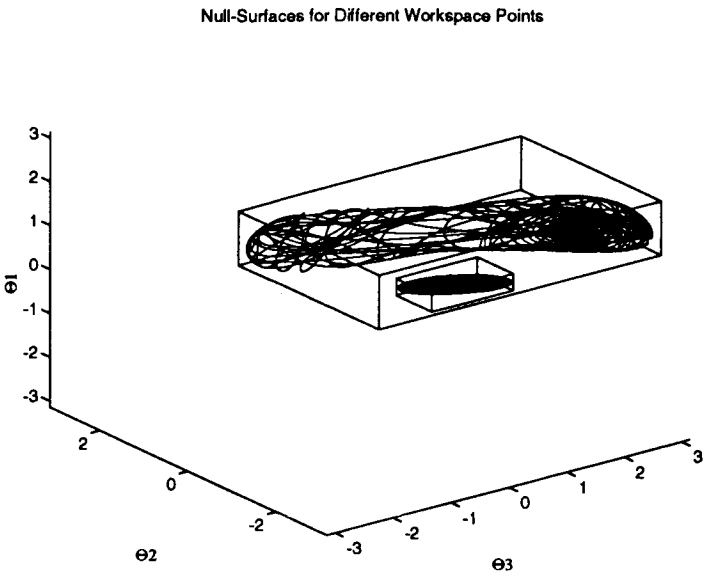


Fig. 5. The set of joint configurations for a kinematically redundant manipulator that yield identical end-effector positions is a surface in an n-dimensional space. A spiral traced out on the tangent plane defined by the null-vectors of the manipulator's Jacobian reveals the shape of the self-motion surface for any given point. Here, two distinct points are shown, one having a large self-motion surface, and one with a small one as indicated by their bounding boxes.

Fig. 4. 3D Slice of the spiral traced out on the self-motion surface in joint space for a PUMA 560 robot used only for positioning.

box. This is evident since regardless of which joint fails, by definition there exists at least one configuration on the self-motion manifold associated with x^* that corresponds to the joint value at which the joint failed. Therefore, the problem of maintaining the fault tolerance of a given critical location reduces to that of maintaining joint limits specified by the bounding box of the self-motion manifold for that location. This problem was first solved in [8] by using (4) and selecting z to result in motion away from the joint limits. The vector z may be computed by combining smooth functions so that the joint limits only effect the manipulators motion when it is near the constraint boundaries [10].

To maintain a high degree of fault tolerance, one would like to locate critical task points in locations where the self-motion manifold bounds are large. For instance jigs and fixtures in general should not be placed near the workspace boundaries since joint failures will render such regions unreachable. Although the tedious chore of measuring the size of the self-motion manifolds throughout the workspace could be done off-line, it has been found that the local measure of fault tolerance, $kfm(\theta)$, is a good indicator of size of the self-motion manifolds.

To insure that a task defined by a sequence of critical points may be performed regardless of joint failures, each point must be analyzed, the associated range of its self-motion surface determined, and then the intersection of the ranges for each point computed to determine the required joint constraints (see Fig. 5). Finally, it must be

verified that the manipulator is able to reach each critical point while maintaining these constraints.

In summary, the following procedure is used to guarantee the failure tolerance of a redundant manipulator with respect to a critical task. First, the workspace is analyzed using the local failure tolerance measure (5). Second, critical task are placed in regions of the workspace that have high values of local failure tolerance. Third, the bounding boxes for the self-motion surfaces associated with each critical location are determined using the procedures outlined in section II. Fourth, the intersection of the bounding boxes is calculated to determine the required constraints. Fifth, each critical workspace point is checked to determine if the manipulator is capable of positioning it's end-effector at the desired location while maintaining the constraints imposed by the intersection of all bounding boxes. Finally, (4) is used with the joint limit constraints to insure the failure tolerance of the manipulator for the specified task.

IV. Conclusions

This paper has developed a method for insuring the failure tolerance of kinematically redundant manipulators. In this work, a redundant manipulator is considered to be fault tolerant with respect to a given task if it is guaranteed to be capable of performing the task after any one of its joints has failed and is locked in place. Methods were developed for analyzing the manipulator's workspace to find regions which are inherently suitable for critical task due to their relatively high level of failure tolerance. Then, the required constraints were imposed on the range of motion of the manipulator to guarantee that a given task is completable regardless of arbitrary joint failures.

References

[1] J. W. Burdick, "On the inverse kinematics of redundant manipulators: Characterization of the self-motion manifolds", *in Proc. 1989 IEEE International Conference on Robotics and Automation*, pp. 264–270, (Scottsdale, AZ), May 14-18 1989.

[2] B. Christensen, W. Drotning, and S. Thunborg, "Model-based, sensor-directed remediation of underground storage tanks", *Journal of Robotic Systems*, vol. 9, pp. 145–159, 1992.

[3] R. Colbaugh and M. Jamshidi, "Robot manipulator control for hazardous waste-handling applications", *Journal of Robotic Systems*, vol. 9, pp. 215–250, 1992.

[4] D. DeMers and K. Kreutz-Delgado, "Issues in learning global properties of the robot kinematic mapping", *in Proc. 1993 IEEE International Conference on Robotics and Automation*, pp. 205–212, (Atlanta, Georgia), May 2-6 1993.

[5] C. A. Klein and B. E. Blaho, "Dexterity measures for the design and control of kinematically redundant manipulators", *International Journal of Robotics Research*, vol. 6, pp. 72–83, Summer 1987.

[6] C. A. Klein and C. H. Huang, "Review of pseudoinverse control for use with kinematically redundant manipulators", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, pp. 245–250, March/April 1983.

[7] C. L. Lewis and A. A. Maciejewski, "Dexterity optimization of kinematically redundant robots in the presence of faults", to appear in *Computers and Electrical Engineering*, 1993.

[8] A. Liégeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-7, pp. 868–871, December 1977.

[9] A. A. Maciejewski, "Kinetic limitations on the use of redundancy in robotic manipulators", *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 205–210, April 1991.

[10] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments", *International Journal of Robotics Research*, vol. 4, pp. 109–117, Fall 1985.

[11] A. A. Maciejewski and C. A. Klein, "The singular value decomposition: Computation and applications to robotics", *International Journal of Robotics Research*, vol. 8, pp. 63–79, December 1989.

[12] C. J. J. Paredis and P. K. Khosla, "Kinematic design of fault tolerant manipulators", to appear in *Computers and Electrical Engineering*, 1993.

[13] M. L. Visinsky, I. D. Walker, and J. R. Cavallaro, "Layered dynamic fault detection and tolerance for robots", *in Proc. 1993 IEEE International Conference on Robotics and Automation*, pp. 180–187, (Atlanta, Georgia), May 2-6 1993.

AIAA-94-1297-CP

UM-PRS: AN IMPLEMENTATION OF THE PROCEDURAL REASONING SYSTEM FOR MULTIROBOT APPLICATIONS

Jaeho Lee, Marcus J. Huber, Edmund H. Durfee, Patrick G. Kenny

Artificial Intelligence Laboratory
The University of Michigan
Ann Arbor, Michigan 48109–2110

Abstract

The Procedural Reasoning System (PRS) is a general purpose reasoning system that is particularly suited for use in domains in which there are predetermined procedures for handling the situations that might arise. We have just completed an implementation of PRS written in C++, which we call the University of Michigan Procedural Reasoning System (UM-PRS). In this paper, we show how UM-PRS provides a critical level of representation for robotic applications in unpredictable domains, because it allows robotic vehicles to pursue long-term goals by adopting pieces of relevant procedures depending on the changing context, rather than having to blindly follow a prearranged plan. Specifically, UM-PRS has been used to control a real outdoor vehicle that changes its behavior based on what it sees in its environment. In turn, this provides the substrate for coordinating multiple robotic vehicles, allowing them to represent joint procedures and to infer each others plans through observation.

Introduction

We have been involved in a project which will, at its terminus, result in a team of robotic vehicles that are capable of autonomously working together while performing reconnaissance and other militarily relevant tasks. High–level plans for these vehicles will typically be in the form of mission plans and annotated maps. A mission plan is a declarative representation of the vehicles' major goals. Based on this, a human annotates a map showing topographical and strategic features to indicate where along planned routes some changes in each robotic vehicle's behaviors must be made (turned on, turned off, or parameters modified). Thus, the annotated map represents a "program" to be executed by the vehicles, where crossing certain spatial lines trigger a vehicle to execute the next step of its program.

The annotated map representation[8] is incomplete in that it lacks the richness required for robotic control in unpredictable, dynamic environments. Blindly following the preprogrammed sequence of behaviors might be hazardous (if the context in which the sequence was formed changes) or impossible (if the vehicle loses track of its position on the map or if its control is temporarily taken over by a human). Therefore, it is important that the map and mission plan be accompanied by more general knowledge about why certain actions are being taken and in what context. Military doctrine, as laid out in documents such as field manuals, contains large numbers of standard operating procedures (SOPs) that should be selectively invoked as conditions and objectives change. In fact, an annotated map for a mission plan typically represents a particular sequence of SOPs that are expected to be useful. For a flexible, autonomous system to succeed, however, the suite of SOPs must be available to the system at runtime.

The Procedural Reasoning System[4–6] is a general purpose reasoning system particularly suited for use in domains in which there are predetermined procedures for handling the situations that might arise. This makes it very applicable in domains such as that discussed above. However, until recently, there has not been an implementation of PRS that is freely available for use. We have just completed an implementation of PRS written in C++, which we call the University of Michigan Procedural Reasoning System (UM-PRS).

In this paper we discuss the details of our initial implementation. Our implementation is novel in terms of both knowledge representations and control structures all written in C++ to meet the needs of efficient real-time robotic control. First, we briefly introduce the general concepts of procedural reasoning systems, the specific representations and the interpreter of the initial UM-PRS version. Second, we illustrate how UM-PRS serves as an important intermediate level of representation and reasoning between the high-level mission plan and the executable annotated map, and how it can interface with these levels. We also illustrate how the flexibility provided by UM-PRS allows autonomous responses beyond those permitted by annotated maps alone. Third, we briefly describe how UM-PRS has been used, to date, in the dynamic control of a real outdoor vehicle and how UM-PRS provides a basis for multi-vehicle coordination, both in terms of allowing the automated formation of belief networks that a vehicle can use to infer the plans of others through observation, and in terms of representing the collective activities of vehicles and the

This research was sponsored in part by ARPA under contract DAAE-07-92-C-R012.

Copyright © 1994 by Jaeho Lee. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

roles that they can play. Finally, we summarize the current status of UM-PRS and the ongoing improvements that we are making in order to realize the full, flexible autonomous capabilities in our multivehicle system.

Procedural Reasoning System

Developing reasoning systems that can reason and plan in continuously changing environments in real-time is emerging as an important area of application of Artificial Intelligence. In this section, we describe basic features of the Procedural Reasoning System (PRS) that motivated us to adopt PRS as a conceptual framework for our system.

The Procedural Reasoning System[4-6] is a general-purpose reasoning system, integrating traditional goal-directed reasoning and reactive behavior. Because most traditional deliberative planning systems formulate an entire course of action before starting execution of a plan, these systems are brittle to the extent that features of the world or consequences of actions might be uncertain. In contrast, the Procedural Reasoning System continuously tests its decisions (both high- and low-level) against its changing knowledge about the world, and can redirect the choices of actions dynamically while remaining purposeful to the extent of the unexpected changes to the environment.

PRS thus is not a planning system in the traditional AI sense, in that PRS does *not* concentrate on searching for sequences of primitive actions that lead to specific goals. Instead, PRS is a plan execution system: it assumes that it already has "plans" (procedures) for achieving various goals in various contexts; however, it might string together actions in unexpected ways as it dynamically chooses among procedures and subprocedures in a changing environment.

Typically, accomplishing a mission in a military setting is much more similar to the plan execution activities of PRS than to the activities of traditional planning systems. Procedures for military tasks such as reconnaissance are developed and learned off-line,[2] and training involves mastering the selection and execution of predefined procedures. These procedures may contain knowledge about both cognitive (such as situation assessment) and physical actions, and they can be arbitrarily complex. Often, a "step" in one procedure (such as "move to assembly area") might itself correspond to several sub-procedures, each appropriate in different contexts.

PRS is conceptually geared for representing precisely this kind of procedural information. Several features that make PRS particularly powerful as a situated reasoning system[6] are as follows:

- The semantics of its plan (procedure) representation, which is important for verification and maintenance.
- Its ability to expand and act on partial plans.
- Its ability to pursue goal-directed tasks while being responsive to changing patterns of events in bounded time.

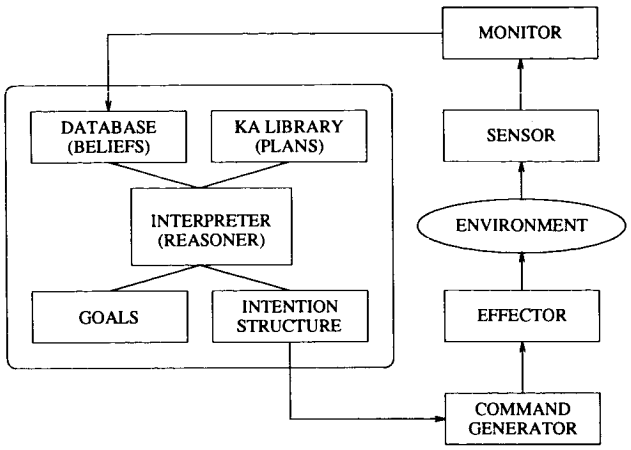


Figure 1: PRS System Structure[3]

- Its facilities for managing multiple tasks in real time.
- Its default mechanisms for handling the environment's stringent real-time demands.
- Its metalevel (or reflexive) reasoning capabilities.

PRS consists of (1) a *database* (called World Model) containing current *beliefs* or facts about the world; (2) a set of current *goals* to be realized; (3) a set of *plans* (called Knowledge Areas) describing how certain sequences of actions and tests may be performed to achieve given goals or to react to particular situations; and (4) an *intention structure* containing those plans that have been chosen for eventual execution (Figure 1). An *interpreter* (or *reasoning mechanism*) manipulates these components, selecting appropriate plans based on the system's beliefs and goals, placing those selected on the intention structure, and executing them.[3]

The system interacts with its environment, including other systems, through its database (which acquires new beliefs in response to changes in the environment) and through the actions that it performs as it carries out its intentions.[3]

UM-PRS

In our particular implementation of PRS, we have been concerned to a large extent with the specific requirements of the military task domain we have outlined, and which we will discuss further below. Thus, whereas some versions of PRS have been developed (typically in Lisp) to be extremely general, our goal has been to identify the crucial features of PRS for our application domain, and to implement them in a robust way in C++. Some of the design and implementation decisions that we have made follow.

World Model

In our implementation, WM is a database of facts which are represented as relations. A relation has a name and a variable number of fields. Initial facts are asserted at the beginning of a UM-PRS program by the user, and other facts can be either asserted or retracted by the KAs, which will be explained below.

Knowledge Areas

A Knowledge Area (KA) is a declarative procedure specification of how to satisfy a system goal or query. It consists of the *purpose* (a goal, query, test, or world model assertion or retraction) for executing the KA, the *context* in which the KA is applicable, a graphical network called the *body* which specifies what is required to satisfy the purpose in terms of primitive functions, subgoals, conditional branches, etc., and a symbol table which holds values for variables when a KA is instantiated for a specific situation. The context consists of a mixed sequence of patterns to be matched against the WM and expressions to be satisfied using the variable bindings generated during the matching.

A SOAK (Set Of Applicable KAs) is a collection of KAs which have been instantiated to achieve a goal (purpose) that has just been activated. Each KA in the SOAK is applicable to the specific situation, as one role of the context is to filter out KAs that are not relevant to a particular situation.

The body describes the procedure's steps, consisting of a network of actions. The body can be viewed as a plan schema. The schema is instantiated with the bindings which are generated when the purpose and the context of the KA are checked during SOAK generation.

Actions

Actions are the arcs in a KA body that constitute either primitive actions or subgoals to achieve. A "primitive" action is a behavior or activity that can be executed directly. Any other type of action represents a) a goal that needs achievement, maintenance, or to be waited upon b) a query, c) a test, or d) an assertion or retraction of world model information. Actions are represented by a base class that holds information regarding the KA in which the action is found and the action name. Each of the types of actions are represented by a derived class that maintains such information as function pointers (for a primitive action), the expression to evaluate (for a test), a goal or query expression, or a world model relation to assert or retract.

Goals

Goals in UM-PRS are the world states that the system is trying to bring about (or maintain, etc.). A goal can be either a top–level goal, which controls the system's highest order behavior, or a subgoal activated by the execution of a KA arc.

Intention Structure

The intention structure acts as the run–time stack for the system. It keeps track of the progress of each high–level goal and all of the subgoals. The intention structure suspends, resumes, cancels, and proceeds with execution of goals in much the same way as an operating system. The intention structure maintains information about what KAs are currently active, as well as what actions in each KA are to be executed next. As there are conditional branches in a KA, the intention structure must also maintain information regarding the success or failure of branches.

The Interpreter

The UM-PRS interpreter is similar to the interpreter described for PRS: It is what controls the execution of the entire system. Whenever there is new or changed information in the world model or goal list, the interpreter determines a new SOAK. From this SOAK is selected the most appropriate KA, which is placed in the intention structure. When there are no SOAKs being generated, the interpreter checks the intention structure for the currently active KAs and executes the next primitive action. If this action changes the goal list (by creating a subgoal or by satisfying a goal) or world model, a new SOAK is created and the cycle starts over. If a new SOAK is not created, then the next arc in a leaf-level KA is executed.

With this implementation, the interpreter facilitates switching to more important goals according to the situation. This implementation also stays committed to one method of achieving a goal by not reconsidering alternatives unless the current method fails. The UM-PRS interpreter is different from the PRS interpreter in that there is currently no metalevel control, although we plan on adding that in the near future as we enrich the set of KAs such that we could have many KAs applicable in overlapping situations.

Example

We began by briefly describing the incompleteness of the annotated map representation[8] in unpredictable, dynamic environments. In this section, we show examples of using both the annotated map representation and the UM-PRS system. We first show a clear correspondence between the annotated map representation and the UM-PRS representation when geographical events are the main triggers of the actions. In the second example, we show the limitation of the annotated map representation and, in contrast, the richness of UM-PRS when general (non-geographical) events trigger actions.

Figure 2 is an example annotated map representation of a simple scenario of getting to an observation point from the assembly area using road following and STRIPE (a waypoint following method) alternatively. The actions to be taken are annotated along the circles in the map. Figure 3 is an example UM-PRS knowledge area that generalizes the procedure on the annotated map. The actions in the KA body roughly correspond to the sequence of actions represented in the annotated map, and the KA representation (context and body) is somewhat simplified to highlight the correspondence between the two representations. A full detailed working example is presented in the Appendix.

An important advantage to using the procedural representation over the annotated map representation is that the correspondence between mission objectives and map markings is made explicit. That is, with an annotated map, the (human) mission planner has a

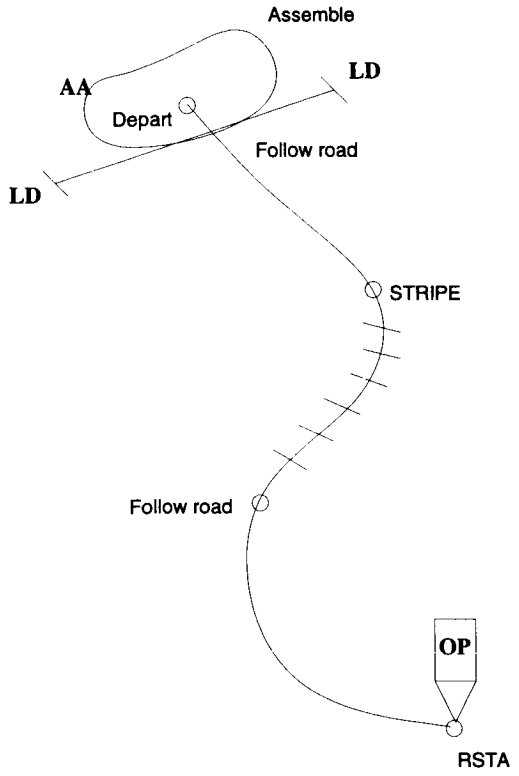


Figure 2: Annotated map

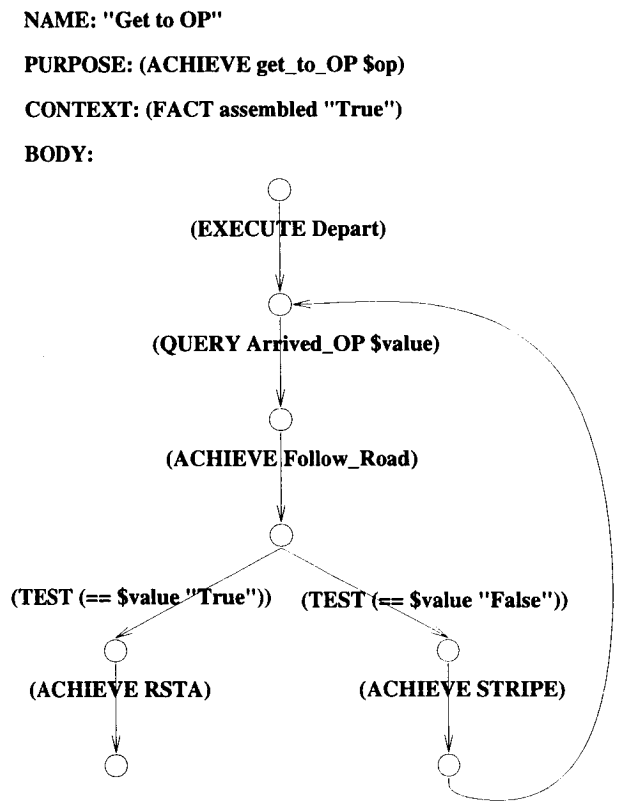


Figure 3: UM-PRS KA

mission in mind, along with an operating procedure to accomplish it. The overall operating procedure is not explicitly represented anywhere, but only the steps are given in the annotations. However, the human interface using a UM-PRS-based system can be quite different. The human can specify an objective, and UM-PRS will retrieve appropriate KAs. In order to instantiate those KAs, UM-PRS must bind variables to values, and those values could include geographical information (where the assembly area is, or which road to follow). Thus, while the user will certainly still point to locations and regions on a map, he or she will do so in response to the needs of the explicitly-represented, standard operating procedures currently being elaborated by UM-PRS.

The second example scenario is shown in Figure 4. The vehicle starts out by issuing a command to start the road-following behavior. This moves the vehicle forward until it reaches a cone or goes past a maximum allowed distance. If it passes the max distance without seeing the cone, the vehicle stops and the demo is done. If the vehicle detects the cone, it approaches the cone and starts off-road behavior until it sees that it has reached the end point. When the vehicle has reached the end point, the demo is done.

The idea of the demo is to show that UM-PRS can be used to represent conditional actions based on non-geographical events such as cone detection. The cone can be placed anywhere along the road, or there may be no cone at all. Such non-geographical events are hard to annotate in a map. The full KA description of this demo is presented in the Appendix. Note that the general capability of pattern-directed invocation of UM-PRS makes it possible to represent high-level choices of very different behaviors as well as simple non-geometric events. So, for example, representing procedures for context changes such as "running for cover" after having "been discovered by enemy" is easy to represent in UM-PRS, rather than cluttering up regions of the map with annotations.

Experiments

In this paper, we have described our implementation of UM-PRS. Currently, we are experimenting with using UM-PRS for robotic control of two indoor mobile robots, and also of an outdoor robotic vehicle (Figure 5). The scenario being used for our experiments is a reconnaissance task in a military domain. Typically, the means of satisfying the task's goals are described in terms of procedures to follow in specific situations. These procedures correspond exactly to KAs, with the context and the purpose specific to the situation in which it is applicable.

One thing we have to note is that all the actions described in the KA body should eventually map down to primitive actions (C or C++ functions) which are directly executable on the real robot or vehicle. Since our implementation is done in C++, the interface between KA actions and real primitive control functions is very natural and efficient.

UM-PRS: Toward Multi-vehicle Coordination

Many tasks of interest, particularly in the military domain, require the concerted efforts of several play-

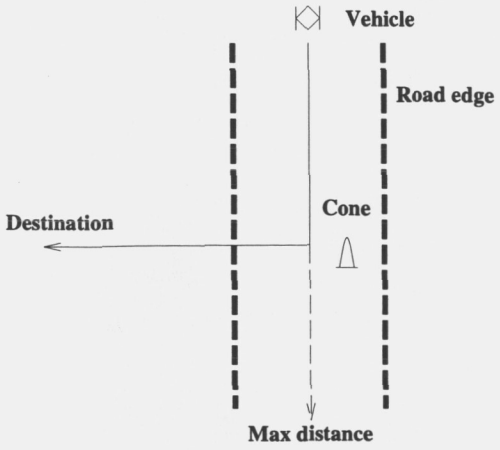


Figure 4: Cone Demo

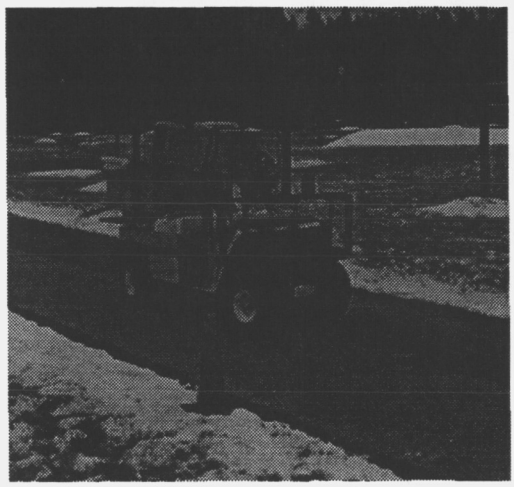


Figure 5: Vehicle and Cone

ers. In other words, they require teamwork. Recently, Georgeff's colleagues[7] have explored extensions to the procedural representation to model different "roles" played in team procedures, and have developed algorithms for assigning roles to potential team members. These capabilities need to be incorporated into UM-PRS to capture the notion of roles in military procedures, such as the roles of "bounder" and "overwatcher" in a bounding-overwatch procedure. Since, in such a procedure, the vehicles take turns watching and moving as they leapfrog across an area, the roles will be assigned and reassigned dynamically in the course of the procedure.

Having each adopted its role in a shared, team procedure, a vehicle can then work to achieve its goals in the procedure. However, since how it chooses to achieve its goals can impact the choices available to other vehicles in how they achieve their goals, some additional coordination is often needed. In essence, while the vehicles might commit to a team plan at an abstract level, they also might have to commit ahead of time to how they might (or might not) elaborate their plans into detailed actions. Anticipating and making necessary commitments ahead of time (before they move into the field where communication is more risky and error prone) should be done, but overcommitment should be avoided lest the vehicles commit to specific courses of action that they later find to be suboptimal or even ineffective. Thus, while the conceptual framework of PRS emphasizes delayed commitment to action until the action must be taken, coordination requires some degree of commitment to the future. Extending UM-PRS to provide this capability is one of our ongoing efforts.

Also, once in the field, some coordination might be needed, and is typically done through communicating about plans, goals, or beliefs about the world. Each of the involved vehicles can reason about this information in order to detect possible conflicts, improvements, synergies, etc. As mentioned before, explicit communication may not always be possible, however, due to such situations as hostile vehicles in the vicinity, environmental noise, and broken equipment. Plan recognition is the process of inferring the same information (the motivating goals or beliefs of a vehicle) based upon sensory observation of that vehicle's actions. Once the plans have been inferred, the same reasoning process can be used to make decisions regarding coordination.

One of the most significant issues that arises is that plans of the robotic vehicles will be in the form of UM-PRS Knowledge Areas, which are not conducive to performing plan recognition. In response to this, we are developing a system that will automatically convert the plans of the other team vehicles into a representation amenable to plan recognition, namely belief networks. Belief networks, also called Bayesian networks,[1] provide the framework and mechanisms for performing probabilistic reasoning about the relationship between the observations of a vehicle's actions and the vehicle's reasons (plans, goals, etc.) for performing that action. While manual construction of belief networks to represent the UM-PRS plans can be done, it is extremely time consuming and subject to variability. When considering the large number of possible KAs for complex tasks, and hence the large number of possible plans that might be created and executed, manual conversion of the plans becomes impractical. By providing an automated system, this process can be performed quickly, efficiently, and without variation.

Conclusions

The representation and control scheme that we have implemented has proven to be sufficiently powerful for planning and execution in procedurally rich domains, specifically in a robotic reconnaissance task. Our experiments have shown the initial implementation to reactively switch between goals, while remaining committed to the current method of achieving each goal. We are actively working on many extensions to the initial implementation (such as metalevel control, and coordination mechanisms, as outlined above) that will make it even more powerful and flexible.

Appendix: Example system

The example system included here as a demo for UM–PRS is an early version of a KA library and primitive functions developed to demonstrate the applicability of UM–PRS to mobile robot tasks.

The idea of the demo is to show that a planner can be used as a triggering device to enable and change vehicle behaviors.

- The vehicle starts out by issuing a command to start the YARF road following behavior. This moves the vehicle forward until it reaches a cone or goes past a maximum allowed distance.
- UM-PRS will wait until the vehicle is stopped (0 = not stopped, 1 = stopped). UM-PRS then will query to see if the vehicle has seen the cone or passed the max distance.
- If it passes the max distance, then the vehicle stops and the demo is done.
- If the vehicle detects the cone, then the vehicle stops and waits for the next behavior.
- If the vehicle sees the cone, then UM-PRS will issue an Approach Cone behavior. UM-PRS will then wait until the vehicle has stopped and check if it has reached the cone.
- If the vehicle has reached the cone, then UM-PRS will issue an Off Road behavior. UM-PRS will wait until the vehicle has stopped and reached the end point.
- When the vehicle has reached the end point, then the demo is done.

KA code

```
GOALS:
        (ACHIEVE cone_demo)

FACTS:
        (vehicle_status "True")
        (demo_done "False")
        (cone_found "False")
        (cone_reached "False")
        (vehicle_reached "False")
        (vehicle_maxdist "False")
        (vehicle_stopped "True")


//-------------------------------------
// KA 1
//-------------------------------------
KA {
NAME:
 "complete cone demo"

DOCUMENTATION:
 "This is the main KA
  that will start the cone demo"

PURPOSE:
 (ACHIEVE cone_demo)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT demo_done "False")

BODY:
 (1 (ACHIEVE vehicle_initialized) 2)
 (2 (ACHIEVE road_scouted) 3)
}

//-------------------------------------
// KA 2
//-------------------------------------

KA {
NAME:
 "initialized vehicle"

DOCUMENTATION:
 "initialized all the vehicle controls"

PURPOSE:
 (ACHIEVE vehicle_initialized)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT demo_done "False")

BODY:
 (99 (EXECUTE init_database 2) 1)
 (1  (EXECUTE home_robot $x $y $to) 2)
 (2  (ASSERT vehicle_initalized)  3)
 (3  (ASSERT YARF              2)  4)
 (4  (ASSERT APPROACHCONE      4)  5)
 (5  (ASSERT OFFROAD           8)  6)
 (6  (ASSERT CHECKVEHICLE     16)  7)
 (7  (ASSERT STOPPED           2)  8)
 (8  (ASSERT CONEFOUND         4)  9)
 (9  (ASSERT MAXDIST           8) 10)
 (10 (ASSERT READCHEDCONE     16) 11)
 (11 (ASSERT READCHEDVEHICLE 32) 12)
 (12 (ASSERT VEHICLESTATUS   64) 13)
}

//-------------------------------------
// KA 3
//-------------------------------------
KA {
NAME:
 "road scouted"

DOCUMENTATION:
 "This KA will scout out a road,
  as per the scenario plan"

PURPOSE:
 (ACHIEVE road_scouted)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT demo_done "False")

BODY:
 (1 (ACHIEVE road_followed_until_cone) 2)
 (OR
   ((2 (FACT cone_found $value) 3)
    (3 (TEST (== $value "True")) 5)
    (5 (ACHIEVE cone_approached) 6)
    (6 (FACT cone_reached $value) 7)
    (7 (TEST (== $value "True")) 8)
    (8 (ACHIEVE traveled_off_road) 9)
```

```
    (9 (FACT vehicle_reached $value) 10)
    (10 (TEST (== $value "True")) 11)
    (11 (ASSERT demo_done "True") 12))
   ((2 (FACT vehicle_maxdist $value) 20)
    (20 (TEST (== $value "True")) 21)
    (21 (ASSERT demo_done "True") 12)))
}

//-------------------------------------
// KA 4
//-------------------------------------
KA {
NAME:
 "road_followed_until_cone"

DOCUMENTATION:
 "This KA will follow the road
  until the vehicle sees a cone
  or passes the turn off point"

PURPOSE:
 (ACHIEVE road_followed_until_cone)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT cone_found "False")
 (FACT vehicle_maxdist "False")
 (FACT YARF $YARF)
 (FACT STOPPED $STOPPED)
 (FACT CONEFOUND $CONEFOUND)
 (FACT MAXDIST $MAXDIST)

BODY:
 (1 (EXECUTE start_behavior $YARF) 2)
 (2 (EXECUTE check_behavior
            $STOPPED
            $vehicle_stopped) 3)
 (3 (FACT vehicle_stopped $value) 4)
 (OR
   ((4 (TEST (== $value "False")) LOOP 2))
   ((4 (TEST (== $value "True")) 5)
    (5 (EXECUTE check_behavior
              $CONEFOUND
              $cone_found) 6)
    (6 (ASSERT  cone_found
              $cone_found) 7)
    (7 (EXECUTE check_behavior
              $MAXDIST
              $vehicle_maxdist) 8)
    (8 (ASSERT  vehicle_maxdist
              $vehicle_maxdist) 9)))

/*
  start YARF behavior
  while (not done)
   if (vehicle stopped)
         done = true
  if (vehicle max distance)
     stop all,
     we are done with demo,
     mission was not accomplished
  else if (cone found)
     assert found cone
*/
}

//-------------------------------------
// KA 5
//-------------------------------------
KA {
NAME:
 "cone_approached"

DOCUMENTATION:
 "When the vehicle sees the cone,
       it approaches it"

PURPOSE:
 (ACHIEVE cone_approached)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT cone_reached "False")
 (FACT APPROACHCONE $APPROACHCONE)
 (FACT STOPPED $STOPPED)
 (FACT REACHEDCONE $REACHEDCONE)

BODY:
 (1 (EXECUTE start_behavior
             $APPROACHCONE) 2)
 (2 (EXECUTE check_behavior
             $STOPPED
             $vehicle_stopped) 3)
 (3 (FACT vehicle_stopped $value) 4)
 (OR
   ((4 (TEST (== $value "False")) LOOP 2))
   ((4 (TEST (== $value "True")) 5)
    (5 (EXECUTE check_behavior
              $REACHEDCONE
              $cone_reached) 6)
    (6 (ASSERT  cone_reached
              $cone_reached) 7)))

/*
 start approach cone behavior
 while (not done)
   if (vehicle stopped)
      done = true
   if (reached cone)
      assert at cone
*/
}

//-------------------------------------
// KA 6
//-------------------------------------
KA {
NAME:
 "traveled_off_road"

DOCUMENTATION:
 "When the vehicle is at the cone,
  it does some off roading"

PURPOSE:
 (ACHIEVE traveled_off_road)

CONTEXT:
 (FACT vehicle_status "True")
 (FACT vehicle_reached "False")
 (FACT STOPPED $STOPPED)
 (FACT REACHEDVEHICLE $REACHEDVEHICLE)
 (FACT OFFROAD $OFFROAD)

BODY:
 (1 (EXECUTE start_behavior $OFFROAD) 2)
```

```
 (2 (EXECUTE check_behavior
            $STOPPED $vehicle_stopped) 3)
 (3 (FACT vehicle_stopped $value) 4)
 (OR
   ((4 (TEST (== $value "False")) LOOP 2))
   ((4 (TEST (== $value "True")) 5)
    (5 (EXECUTE check_behavior
                $REACHEDVEHICLE
                $vehicle_reached) 6)
    (6 (ASSERT  vehicle_reached
                $vehicle_reached) 7)))

/*
   start off road behavior
   while (not done)
     if (vehicle stopped)
           done = true
   if (reached vehicle)
     assert at vehicle
*/
}

//-----------------------------------
// KA 7
//-----------------------------------
KA {
NAME:
 "vehicle_status_checked"

DOCUMENTATION:
 "Check to make sure the vehicle is ok
  once in a while"

PURPOSE:
 (ACHIEVE vehicle_status_checked)

CONTEXT:
 (FACT CHECKVEHICLE $CHECKVEHICLE)
 (FACT VEHICLESTATUS $VEHICLESTATUS)

BODY:
 (1 (EXECUTE start_behavior
             $CHECKVEHICLE) 2)
 (2 (EXECUTE check_behavior
             $VEHICLESTATUS
             $vehicle_status) 3)
 (3 (ASSERT vehicle_status
            $vehicle_status) 4)

 /*
    if (vehicle status != OK)
      stop vehicle, stop prs
 */
 }
```

References

[1] Eugene Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, Winter 1991.

[2] Department of the Army, Washington, D.C. *Tank Platoon, FM 17-15*, October 1987.

[3] Michael P. Georgeff and François Félix Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 972–978, Detroit, Michigan, 1989.

[4] Michael P. Georgeff and Amy L. Lansky. Procedural knowledge. *Proceedings of the IEEE*, 74(10):1383–1398, October 1986.

[5] François Félix Ingrand and Michael P. Georgeff. Managing deliberation and reasoning in real-time AI systems. In *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 284–291, San Diego, CA, November 1990.

[6] Francois F. Ingrand, Michael P. Georgeff, and Anand S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):34–44, December 1992.

[7] David Kinny, Magnus Ljungberg, and Anand Rao. Planned team activity. In Amedeo Cesta, Rosaria Conte, and Maria Miceli, editors, *Pre-Proceedings of the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, pages 1–20, Rome, Italy, July 1992.

[8] Charles Thorpe, Martial Hebert, Takeo Kanade, and Steven Shafer. Toward autonomous driving: The CMU navlab. *IEEE Expert*, pages 31–52, August 1991.

CONTROL OF PARALLEL MANIPULATORS USING FORCE FEEDBACK

Prabjot Nanua
University of Houston
Houston, Texas

Abstract

Parallel robotic mechanisms have good accuracy, high stiffness and large payload to weight ratio compared to the traditional serial mechanism. This paper compares two simple constant gain control schemes for a parallel robotic mechanism actuated by hydraulic cylinders. One of the control schemes which will be referred to as a "rate based scheme" uses the position and rate information only for feedback. The other control scheme referred to as the "force based scheme" feeds back the force information also. It is shown that for parallel robots with hydraulic actuators the response of the end-effector can be improved by using the force information from the actuators without adding any extra computational burden. The force based control scheme can also be easily modified to control the forces on the end-effector. The control scheme has been implemented in a computer simulation and the results are presented in the paper.

Introduction

Recently, there has been a lot of activity in the area of parallel mechanisms (Figure 1). Parallel mechanisms are able to overcome many of the shortcomings of serial mechanisms. They have a high stiffness, large payload to weight ratio and good accuracy compared to serial mechanisms. The best known application of parallel mechanism is the Stewart Platform which is used in aircraft simulators. These mechanisms also have potential for application in zero/partial gravity simulators, assembly tasks and precision machining.

Previous research in parallel mechanisms has been focused on the kinematics of the mechanisms. The general kinematic considerations are examined in [3], [4], [7], [13] and [14]. The direct position kinematics of some special parallel mechanisms are given in [5], [9] and [10]. There is little research in the dynamics and controls of parallel mechanisms. Some of the dynamics issues are examined in [15].

An important part of assembly tasks is the control of interaction forces between components. In other applications such as partial/zero gravity simulators, it will be required that the actuators apply constant forces through the center of mass of the end effector. In this application, again the forces on the end effector will have to be controlled. The rate based scheme does not offer any capability for controlling forces. The force based control scheme developed in this paper can easily be modified to control these forces. This ability is crucial to the successful application of parallel mechanisms to assembly tasks.

The parallel mechanism analyzed in this paper is shown in Figure 1. It is a six degree of freedom mechanism. The top member and the base member are connected by six limbs. Each limb is a hydraulic cylinder with universal joints at each end. The piston and the cylinder are allowed to rotate with respect to each other. Thus each limb is a six degree of freedom serial chain with one actuated prismatic joint. The universal joints in the top member are located in a plane and the universal joints in the base member are also located in a single plane.

The first section of the paper examines the linearized dynamic model of the fully parallel mechanism. Using the linearized model, the control schemes are studied. The following section describes the control scheme in detail. The next section deals with the

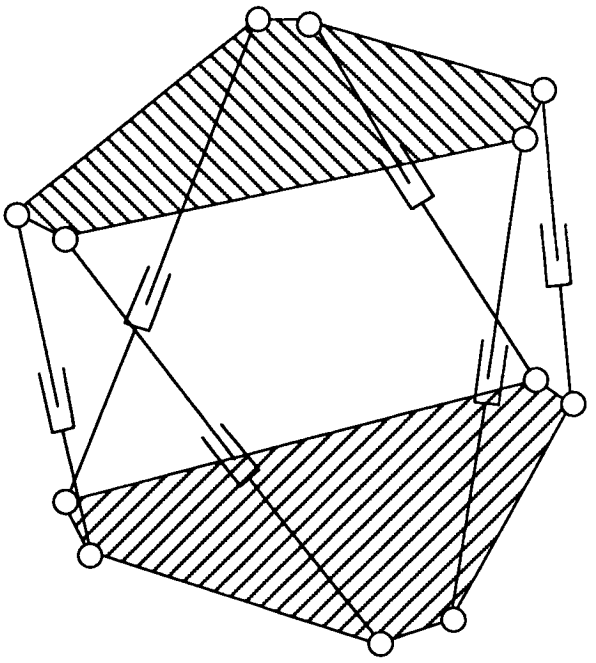


Figure 1. Special Parallel Mechanism (Each limb consists of universal joints at each end and a cylinderical joint between them; the prismatic joint is actuated).

Copyright © American Institute of Aeronautics and Astronautics, Inc., 1976. All rights reserved.

complete dynamic simulation of the parallel mechanism. To simulate the dynamics of the mechanism, it is essential to have an efficient scheme for the direct dynamics problem. This will help in reducing the computational time required for simulation. The models for the hydraulic cylinders and the servolvalves are given in Appendix B. The results are discussed in the last section.

Linearized Rigid Body Model

In fully parallel mechanisms ([14]), the joint rates and the end effector motion are related by the following equation:

$$\dot{\mathbf{L}} = \mathbf{H}^{\mathrm{T}} \begin{bmatrix} \omega \\ \mu \end{bmatrix} \tag{1}$$

where
$\dot{\mathbf{L}}$ = Vector consisting of the joint rates.
$\mathbf{H}$ = 6×6 matrix.
ω = angular velocity of the end effector.
μ = velocity of the point on the end effector coincident with the origin.

The matrix $\mathbf{H}$ is a purely geometric quantity. The columns of the matrix $\mathbf{H}$ are the wrench axis of the joints.

For parallel mechanisms, static force decomposition equation is:

$$\begin{bmatrix} \mathbf{R} \\ \Gamma \end{bmatrix} = \mathbf{HF} \tag{2}$$

where
$\mathbf{R}$ = Forces acting on the end effector.
Γ = Moments acting on the end effector about the origin.
$\mathbf{H}$ = 6×6 matrix
$\mathbf{F}$ = Forces/Torque's at the joints.

It will be assumed without any loss of generality that the origin of the fixed coordinate system is located at the nominal position of the center of mass of the end-effector.

The current approach, based on the rate control scheme, is to control the joint lengths in the parallel mechanism. Using this scheme, current position of the end effector is compared to the desired position. The error in the position is converted to an error in the length of the limbs. The limbs are then individually controlled to eliminate this error. This is achieved by applying forces in the limbs that are proportional to the error in the limb lengths. This is given by the equation:

$$\mathbf{F} = -\mathbf{K}_p\mathbf{L} - \mathbf{K}_v\dot{\mathbf{L}} \tag{3}$$

where
$\mathbf{L}$ = error in the limb lengths.

The equation of motion of the end effector is given by (ignoring the non-linear terms):

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{J} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{p}} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \mathbf{R} \\ \Gamma \end{bmatrix} \tag{4}$$

where
$\mathbf{M}$ = Diagonal matrix with the mass of the end effector as the diagonal term.
$\mathbf{J}$ = Moment of inertia matrix at the current position.
$\mathbf{p}, \theta$ = small errors in position and orientation of end effector.

Using equations (2), (3) and (4), the response equation is (ignoring the actuator model):

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{J} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{p}} \\ \ddot{\theta} \end{bmatrix} + \mathbf{H}\mathbf{K}_v\mathbf{H}^{\mathrm{T}} \begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\theta} \end{bmatrix} + \mathbf{H}\mathbf{K}_p\mathbf{H}^{\mathrm{T}} \begin{bmatrix} \mathbf{p} \\ \theta \end{bmatrix} = \mathbf{0} \tag{5}$$

The gain matrices in the above equation, $\mathbf{K}_p$ and $\mathbf{K}_v$ are diagonal matrices with constant terms (the cylinders are controlled independently with constant gains). Clearly, the response depends on the matrix $\mathbf{H}$, which depends on the position of the end effector. This is undesirable. It is also not obvious from the above equation, whether an increase in the gains would lead to an improvement in the response of the mechanism. Increase in the gains might lead to deterioration in the response due to the structure of the $\mathbf{H}$ matrix. Further, increasing the gains, leads to a shift in all the closed loop poles of the above system. The fast poles as well as the slow poles are moved simultaneously. This leads to a marginal improvement in the response of the system with large changes in the gains. There is also the possibility of exciting the higher order dynamics due to the presence of fast poles (as a result of increase in gains).

As opposed to the above rate based scheme, in the force based control scheme, the force is controlled in each limb. The error in the current position of the end effector is fed back as error in the forces exerted by the limbs. This is given by the equation:

$$\begin{bmatrix} \mathbf{R} \\ \Gamma \end{bmatrix} = -\mathbf{K}_p \begin{bmatrix} \mathbf{p} \\ \theta \end{bmatrix} - \mathbf{K}_v \begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\theta} \end{bmatrix} \tag{6}$$

The force to be produced by the actuators at the limbs is given by (equation (2) and (6)):

$$\mathbf{F} = -\mathbf{H}^{-1}\left(\mathbf{K}_p\begin{bmatrix}\mathbf{p}\\ \theta\end{bmatrix} + \mathbf{K}_v\begin{bmatrix}\dot{\mathbf{p}}\\ \dot{\theta}\end{bmatrix}\right) \tag{7}$$

The response equation is now given by (combining equations (4) and (6)):

$$\begin{bmatrix}\mathbf{M} & \mathbf{0}\\ \mathbf{0} & \mathbf{J}\end{bmatrix}\begin{bmatrix}\ddot{\mathbf{p}}\\ \ddot{\theta}\end{bmatrix} + \mathbf{K}_v\begin{bmatrix}\dot{\mathbf{p}}\\ \dot{\theta}\end{bmatrix} + \mathbf{K}_p\begin{bmatrix}\mathbf{p}\\ \theta\end{bmatrix} = \mathbf{0} \tag{8}$$

In a simple implementation of the above scheme, $\mathbf{K}_p$ and $\mathbf{K}_v$ can be chosen as diagonal matrices with fixed diagonal terms. The response equations are now to a large extent position independent and decoupled. The response does not depend on the **H** matrix. The coupling and the position dependency in the above equation will come from the **J** matrix. This response equation is an improvement over the previous scheme. The closed loop poles can be assigned independently in the above scheme and the response improved without a corresponding large increase in the gains.

The above scheme can also be easily adapted to control the forces on the end effector. The direction in which forces are to be controlled can easily replace the position errors in equation (6). This would effectively control the forces in those directions.

Dynamic Simulation and Control Scheme of the Parallel Mechanism

A rigid body dynamic model was developed for the parallel mechanism for simulation in MATRIXX. Some of the assumptions made for the dynamic model are as follows. The limbs were assumed to be axisymmetric. The friction losses in the spherical and revolute joints were assumed to be negligible. The prismatic joint in the cylinder was assumed to have viscous losses and the damping coefficient was adjusted to obtain a response that closely approximated the response from the actual mechanism. The model for the hydraulic cylinders consists of a leakage term which is assumed to be proportional to the pressure difference in the cylinder (assuming that the leakage flow is laminar). The model for the servovalves was taken from [8].

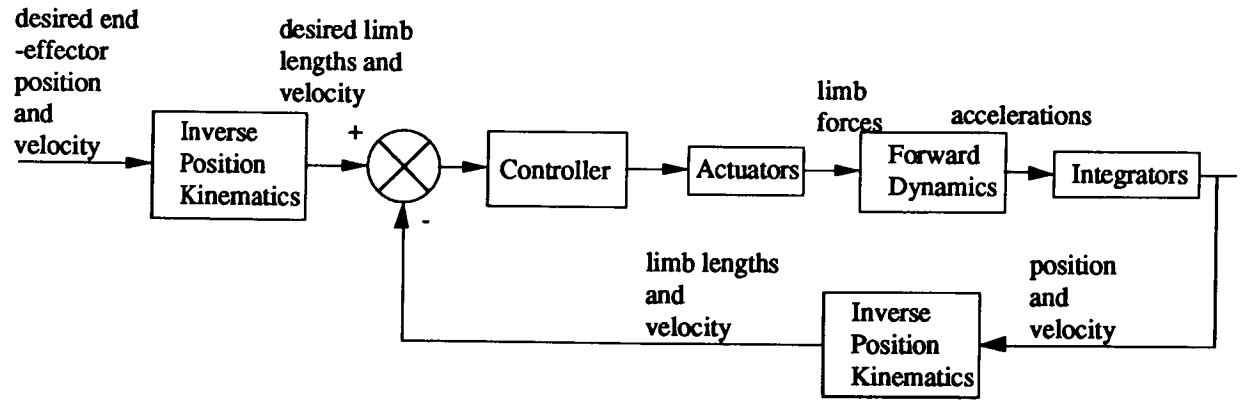


Figure 2. Block Diagram for Rate Based Control Scheme.

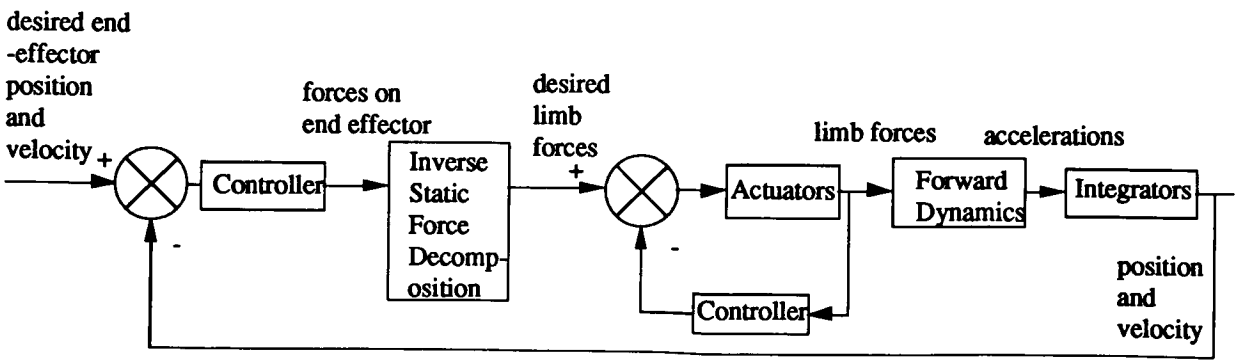


Figure 3. Block Diagram for Force Based Control Scheme.

The block diagram for the rate based control scheme is shown in Figure 2. In this control scheme, the hydraulic cylinders were controlled by dual stage servovalves. Within the servovalve, the spool position is fed back to the electromagnet with a spring. The block diagram for the servolvalve is given in Appendix B. The error in the position and the velocity are fed back to the current in the servovalve. The current in the servovalve is given by:

$$\mathbf{i} = -\mathbf{k_p I e_L} - \mathbf{k_v I \dot{e}_L} \tag{9}$$

where
$\mathbf{i}$ = 6×1 vector.
$\mathbf{k_p}$, $\mathbf{k_v}$ = constants.
$\mathbf{I}$ = 6×6 identity matrix.
$\mathbf{e_L}$ = error in limb lengths.
$\dot{\mathbf{e}}_L$ = error in limb length rates.

A decoupled controller with constant gains is chosen for the controller because the gains are associated with the hydraulic cylinders. Since all the hydraulic cylinders are identical, the gains are kept the same for all the cylinders.

The block diagram for the force based control scheme is shown in Fig. 3. In this control scheme, the cylinder were controlled by a proportional servovalve. It was found that the spool position feedback in the dual stage servovalve inhibited the response of the mechanism. The maximum spool opening from the servovalve was the same in both the valves to ensure that a fair comparison could be made in both the schemes.

The desired force in each of the limbs is given by:

$$\mathbf{F} = -\mathbf{H}^{-1}\left(\mathbf{K_p}\begin{bmatrix}\mathbf{e_p}\\ \mathbf{e_\theta}\end{bmatrix} + \mathbf{K_v}\begin{bmatrix}\dot{\mathbf{e}}_p\\ \dot{\mathbf{e}}_\theta\end{bmatrix}\right) \tag{10}$$

where
$\mathbf{K_p}$, $\mathbf{K_v}$ = Diagonal constant matrices.
$\mathbf{e_p}$ = error in position.
$\dot{\mathbf{e}}_p$ = error in velocity
$\mathbf{e_\theta}$ = error in angular position.
$\dot{\mathbf{e}}_\theta$ = error in angular velocity.

The gain matrices $\mathbf{K_p}$ and $\mathbf{K_v}$ are chosen to be diagonal matrices. At the current location of the end-effector, the off diagonal terms could have been chosen to decouple the system. This decoupling however, would not be possible for the complete workspace due to the change in the inertia matrix.

The error in the desired force in the limbs given by equation (10) and the actual force in the cylinder is fed back to the spool. The position of the spool is then given by:

$$\mathbf{X_s} = \mathbf{k_f e_f} \tag{11}$$

where
$\mathbf{k_f}$ = constant.
$\mathbf{e_f}$ = 6×1 error vector in the forces.

The gain for the force controller are kept the same for all the cylinders, since the cylinders are identical.

Rigid Body Dynamic Model

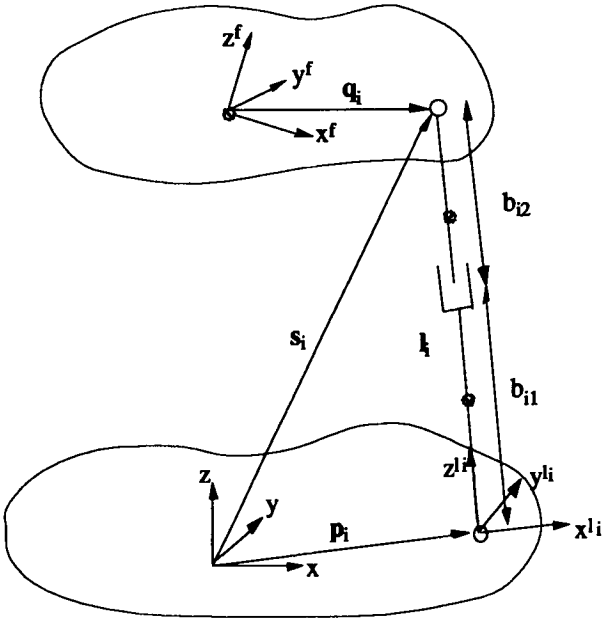


Figure 4. Parallel mechanism with a single limb and the variables.

The six degree of freedom parallel mechanism shown in Figure 1 consists of 13 rigid body elements. The six equations of motion can be obtained for each element, resulting in a large matrix (in this case 78×78 matrix) equation. This equation will be computationally expensive to solve. In the following section, a compact 6×6 matrix equation for the dynamics of the parallel mechanism is obtained. This would result in substantial savings in the computational time for the simulation.

The convention followed in the rest of the paper is as follows. All vectors and matrices are in bold letters. The superscript on the top left hand corner of a vector denotes the coordinate system. A superscript "t" refers to the fixed coordinate system coincident with the coordinate system attached to the top member and

"l_i" refers to the fixed coordinate system coincident to the coordinate system attached to the ith limb. No superscript signifies the fixed world coordinate system attached to the base member.

The equations of motion of the top member are given by:

$$\mathbf{M}^t\mathbf{a} = -\sum_i {}^t\mathbf{F}_{i3} + \mathbf{M}^t\mathbf{g}$$

$$\mathbf{J}^t\alpha = -\sum_i \left({}^t\mathbf{q}_i \times {}^t\mathbf{F}_{i3} + {}^t\mathbf{T}_{i3}\right) + {}^t\omega \times \mathbf{J}^t\omega \qquad (12)$$

where

$$\mathbf{M} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{bmatrix}$$

${}^t\mathbf{g} = {}^t\mathbf{R}_b [0 \quad 0 \quad -g]^T$

g = acceleration due to gravity.

${}^t\mathbf{F}_{i3}$ = force applied by the top member to the ith limb.

${}^t\mathbf{T}_{i3}$ = Torque applied by top member to the ith limb.

${}^t\mathbf{q}_i$ = position of the ith force.

${}^t\omega$ = angular velocity of the top member in the top coordinate frame.

In the above equations, the forces are unknown. These forces are applied to the top member by the limbs. The additional equations will come from the dynamic analysis of the limbs. These dynamic equations will be in the form:

$${}^{l_i}\mathbf{a}_{i3} = \mathbf{D}_i {}^{l_i}\mathbf{F}_{i3} + \mathbf{d}_i \qquad (13)$$

where

$\mathbf{a}_{i3}$ = acceleration of the ith universal joint.

$\mathbf{F}_{i3}$ = force transmitted across the ith universal joint.

The development of the above equations for the parallel mechanism under consideration here are given in Appendix A.

The acceleration of the joint and the center of mass of the top member are related by:

$${}^{l_i}\mathbf{a}_{i3} = {}^{l_i}\mathbf{R}_t \left({}^t\mathbf{a} - \mathbf{qX}\,{}^t\alpha + \mathbf{t}\right) \qquad (14)$$

where

$$\mathbf{qX} = \begin{bmatrix} 0 & -{}^tq_{iz} & {}^tq_{iy} \\ {}^tq_{iz} & 0 & -{}^tq_{ix} \\ -{}^tq_{ix} & {}^tq_{ix} & 0 \end{bmatrix}$$

$$\mathbf{t} = \begin{bmatrix} -{}^tq_{ix}\left({}^t\omega_y^2 + {}^t\omega_z^2\right) + {}^t\omega_x\left({}^t\omega_z\,{}^tq_z + {}^t\omega_y\,{}^tq_y\right) \\ -{}^tq_{iy}\left({}^t\omega_z^2 + {}^t\omega_x^2\right) + {}^t\omega_y\left({}^t\omega_x\,{}^tq_x + {}^t\omega_z\,{}^tq_z\right) \\ -{}^tq_{iz}\left({}^t\omega_x^2 + {}^t\omega_y^2\right) + {}^t\omega_z\left({}^t\omega_y\,{}^tq_y + {}^t\omega_x\,{}^tq_x\right) \end{bmatrix}$$

Combining (13) and (14) and rearranging terms:

$$\begin{bmatrix} \mathbf{I}_{3\times3} & -\mathbf{qX} \end{bmatrix} \begin{bmatrix} {}^t\mathbf{a} \\ {}^t\alpha \end{bmatrix} = {}^t\mathbf{R}_{l_i} \mathbf{D}^{l_i}\mathbf{R}_t\, {}^t\mathbf{F}_{i3} + {}^t\mathbf{R}_{l_i}\mathbf{d}_i - \mathbf{t} \qquad (15)$$

The above constraint equation can be obtained for each limb. This will give 6 such equations.

Another set of constraint equations can be obtained for $\mathbf{T}_{i3}$. The angular accelerations of the upper part of the limb and the top member are related by:

$$\begin{bmatrix} - \\ - \\ {}^{l_i}\alpha_{iz} \end{bmatrix} = {}^{l_i}\mathbf{R}_t \begin{bmatrix} {}^t\alpha_x \\ {}^t\alpha_y \\ {}^t\alpha_z \end{bmatrix} \qquad (16)$$

The top two members of the above vector equation are not important and thus are left blank. This equation can be combined with the last equation in equation set (16) to get:

$${}^t\mathbf{T}_{i3} = -{}^t\mathbf{R}_{l_i} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & J_{i2z} \end{bmatrix} {}^{l_i}\mathbf{R}_t\, {}^t\alpha \qquad (17)$$

The equation of motion of the top member under the influence of forces $\mathbf{F}_{i3}$ are given by (12). Equation (15) can be used to eliminate forces $\mathbf{F}_{i3}$ from equation (12) and equation (17) can be used to eliminate $\mathbf{T}_{i3}$ from equation (2). This leads to:

$$\begin{bmatrix} \mathbf{M} + \sum_{i=1,6} {}^{t}\mathbf{R}_{l_i}\mathbf{D}_i^{-1}{}^{l_i}\mathbf{R}_t & \sum_{i=1,6} q\mathbf{X}^{t}\mathbf{R}_{l_i}\mathbf{D}_i^{-1}{}^{l_i}\mathbf{R}_t \\ \sum_{i=1,6} q\mathbf{X}^{t}\mathbf{R}_{l_i}\mathbf{D}_i^{-1}{}^{l_i}\mathbf{R}_t & \mathbf{J} + \sum_{i=1,6}\left({}^{t}\mathbf{R}_{l_i}\mathbf{J}_{i2z}{}^{l_i}\mathbf{R}_t - q\mathbf{X}^{t}\mathbf{R}_{l_i}\mathbf{D}_i^{-1}{}^{l_i}\mathbf{R}_t q\mathbf{X}\right) \end{bmatrix} \begin{bmatrix} {}^{t}\mathbf{a} \\ {}^{t}\alpha \end{bmatrix} = \begin{bmatrix} \sum_{i=1,6}\left({}^{t}\mathbf{R}_{l_i}\mathbf{D}_i^{-1}\mathbf{d}_i - {}^{t}\mathbf{R}_{l_i}\mathbf{D}_i^{-1}{}^{l_i}\mathbf{R}_t t\right) + \mathbf{M}^{t}\mathbf{g} \\ \sum_{i=1,6}\left(q\mathbf{X}^{t}\mathbf{R}_{l_i}\mathbf{D}_i^{-1}\mathbf{d} - q\mathbf{X}^{t}\mathbf{R}_{l_i}\mathbf{D}_i^{-1}{}^{l_i}\mathbf{R}_t t\right) + {}^{t}\omega \times \mathbf{J}^{t}\omega \end{bmatrix} \tag{18}$$

These equations can be used to solve for the accelerations of the top member. These are a very efficient form of the dynamic equations.

The above accelerations are in the coordinate system fixed to the top member. They can be transformed to the accelerations in the fixed coordinate frame by a simple coordinate transformation. Integrating the accelerations will give the velocity of the top member. The angular velocity of the top member can be used to calculate the euler angle rates using the following equation:

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} = \mathrm{Sec}\beta \begin{bmatrix} \mathrm{Cos}\gamma & \mathrm{Sin}\gamma & 0 \\ -\mathrm{Sin}\gamma\,\mathrm{Cos}\beta & \mathrm{Cos}\gamma\,\mathrm{Cos}\beta & 0 \\ \mathrm{Cos}\gamma\,\mathrm{Sin}\beta & \mathrm{Sin}\gamma\,\mathrm{Sin}\beta & \mathrm{Cos}\beta \end{bmatrix} {}^{t}\omega \tag{19}$$

The above rigid body model was combined with the dynamic models for the hydraulic cylinder and the servolvalves. The details of these models are given in Appendix B.

Discussion of Results

The results from the simulation are shown in Figures (5)-(10). The top member was moved along the z-axis from a initial position of (0,0,140.0) to a final position of (0,0,140.2). while keeping the orientation fixed. The move was commanded at 0.1s from the start of the simulation to allow for the system to attain equilibrium. The gains for the controller were obtained by modifying the gains obtained from a linearized analysis.

The results from the rate based scheme are shown in Figures (5)-(7). The final position is obtained in 0.122s. The move in the z-direction induces oscillation in the x and y directions. If the gains are increased any further, the system will become unstable.

The results from the force based scheme are shown in Figures (8)-(10). The final z-position is reached in 0.058s. This is a 53% improvement in the step response of the mechanism in the z-direction. The gains in this system can be individually adjusted for each direction and thus the response in the x, y and z direction can be individually tailored. There is also a significant steady state error in the response. This is an expected error due to the weight of the top member and the upper part of the limbs. This error can be removed by adding a feedforward term to the controller. The mechanism in the beginning of the move can estimate the weight of the top member and then use this information in the feedforward term.

Conclusions

A simple constant gain force control scheme has been devised for the parallel mechanisms. This scheme is easy to implement and is better suited to parallel mechanisms than rate control schemes. The force control scheme requires the computation of $\mathbf{H}^{-1}$ while the rate control scheme uses the inverse position kinematics. The **H** matrix can be inverted symbolically to save on computational expense.

The preliminary results indicate that the force based control scheme will improve the response over the rate based control scheme. Further work, however is required to reach any comprehensive conclusions. There are important unanswered questions about the effect of the position of the end-effector and the payload mass on the response of the mechanism.

This control scheme can be easily adapted to control forces on the end effector. This capability will be crucial in applications of parallel mechanisms such as assembly tasks or partial/zero gravity simulators. The force control capabilities of the above control scheme also needs further investigation.

Acknowledgments

The author wishes to acknowledge the support of NASA JSC grant no NAG 9-672 for the above research.

References

[1] D'Azzo, J. J., and Houpis, C. H., "Linear Control System Analysis and Design", McGraw Hill Book Co., 1988.

[2] Featherstone, R., "Robot Dynamics Algorithms", Kluwer Academic Publishers, 1987.

[3] Fichter, E. F., "A Stewart Platform - Based Manipulator: General Theory and Practical Construction", International Journal of Robotics Research, vol. 5, No. 2, pp. 157-182, Summer 1986.

[4] Hunt, K. H., "Structural Kinematics of In-Parallel-Actuated Robot-Arms", Trans. ASME J. of Mechanisms, Transmissions, and Automation in Design, Vol 105, pp. 705-712, 1983.

[5] Lee, K. M., and Shah, D. K., "Kinematic Analysis of a Three Degree of Freedom In - Parallel Actuated Manipulator", Proceedings of the 1987 IEEE International Conference on Robotics and Automation, vol. 1, pp 345-350, 1987.

[6] McCallion, H., and Truong, P. D., "The analysis of a Six-Degree-of-Freedom Work Station for Mechanized Assembly", Proceedings of Fifth World Congress for the Theory of Machines and Mechanisms, ASME, pp. 611-616, 1979.

[7] Merlet, J. P., "Parallel Manipulators", Seventh CISM-IFToMM Symposium on theory and practice of Robots and Manipulators, pp. 317-324, Sept. 1988.

[8] Moog, "Type 30 Flow Control Servovalves", Catalog 301 385, Moog Inc.

[9] Nanua, P., and Waldron, K. J., "Direct Kinematic Solution of a Special Parallel Robot Structure", Proceedings of CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators, Cracow, Poland, 1990.

[10] Nanua, P., Waldron, K. J., and Murthy, V., "Direct Kinematic Solution of a Stewart Platform", IEEE Transactions on Robotics and Automation, August 1990.

[11] Potton, S. L., "GEC Advanced Device for Assembly", Proceedings of the CIRP Conference on Assembly and Automation, pp. 126-128, June 1983.

[12] Stewart, D., "A Platform with Six Degrees of Freedom", Proc. Inst. Mech. Engr, 180(1):371-386, 1965.

[13] Sugimoto, K., "Kinematic & Dynamic Analysis of Parallel Manipulators by Means of Motor Algebra," Trans. ASME J. of Mechanisms, Transmissions, and Automation in Design, vol. 109, pp. 3-7, 1987.

[14] Waldron, K. J., and Hunt, K. H., "Series - Parallel Dualities in Actively Coordinated Mechanisms", 4th Int. Symposium on Robotics Research, Santa Cruz, CA, 1987.

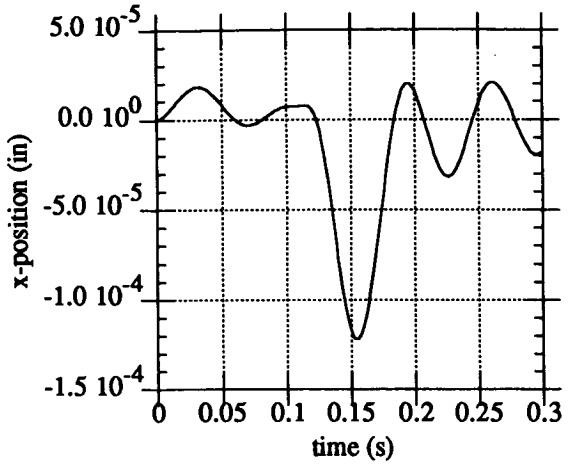


Figure 5. X-Position under rate-based control scheme.

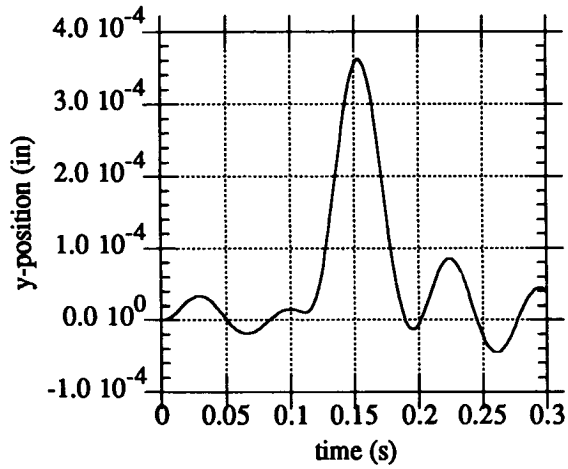


Figure 6. Y-Position under rate-based control scheme.

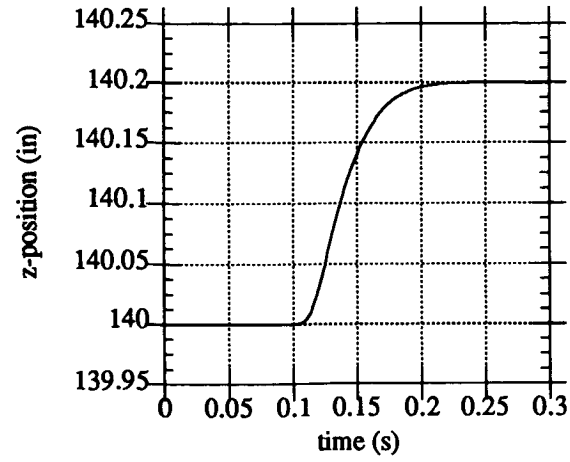


Figure 7. Z-Position under rate-based control scheme.

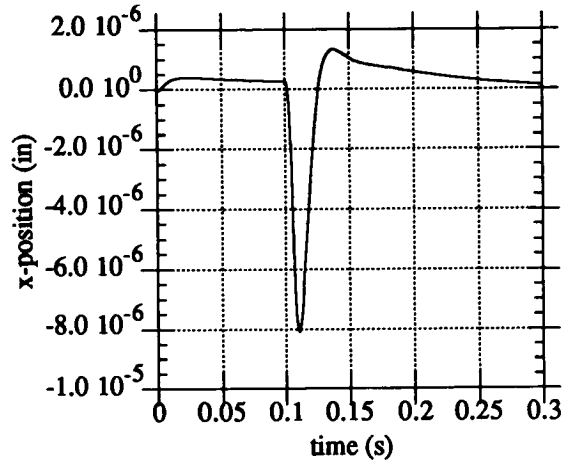


Figure 8. X-Position for force-based control scheme.

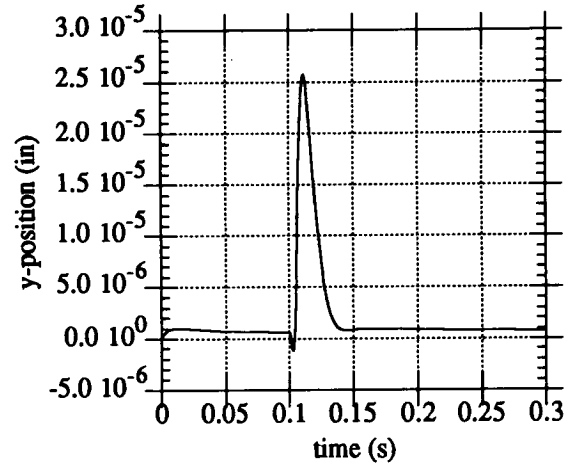


Figure 9. Y-Position for force-based control scheme.

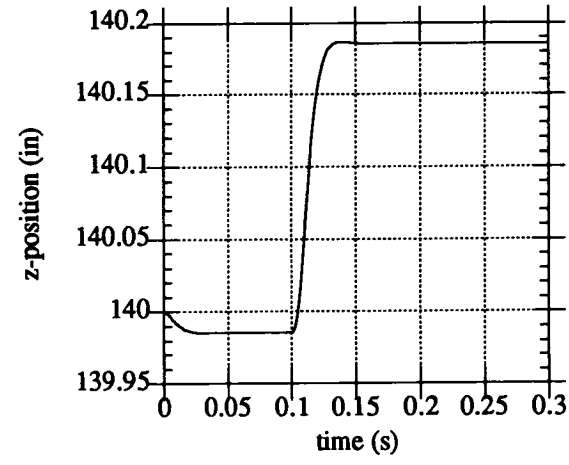


Figure 10. Z-Position for force-based control scheme.

Appendix A

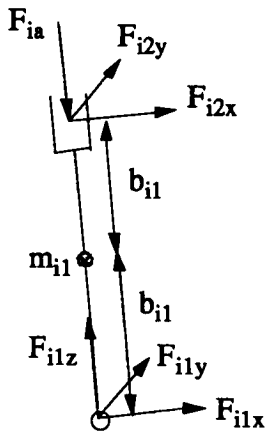


Figure a1. Free body diagram of the lower part of limb.

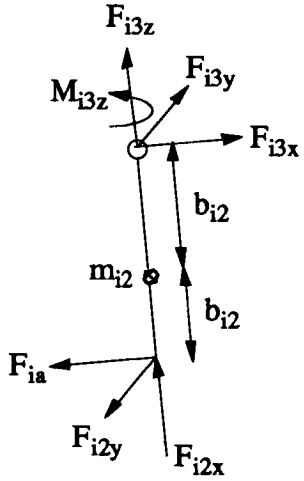


Figure a2. Free body diagram of the upper part of the limb.

Given the position of the coordinate system attached to the top member and its orientation, the position vector of the ith joint on the top member, in the fixed coordinate system can be computed by

$$\mathbf{s_i} = {}^{b}\mathbf{R}_t \, {}^{t}\mathbf{q_i} + \mathbf{r}$$

where

$$ {}^{b}\mathbf{R}_t = \begin{bmatrix} \text{Cos}\gamma\,\text{Cos}\beta & -\text{Sin}\gamma\,\text{Cos}\alpha + \text{Cos}\gamma\,\text{Sin}\beta\,\text{Sin}\alpha & \text{Sin}\gamma\,\text{Sin}\alpha + \text{Cos}\gamma\,\text{Sin}\beta\,\text{Cos}\alpha \\ \text{Sin}\gamma\,\text{Cos}\beta & \text{Cos}\gamma\,\text{Cos}\alpha + \text{Sin}\gamma\,\text{Sin}\beta\,\text{Sin}\alpha & -\text{Cos}\gamma\,\text{Sin}\alpha + \text{Sin}\gamma\,\text{Sin}\beta\,\text{Cos}\alpha \\ -\text{Sin}\beta & \text{Cos}\beta\,\text{Sin}\alpha & \text{Cos}\beta\,\text{Cos}\alpha \end{bmatrix} $$

${}^{t}\mathbf{q_i}$ = position of ith joint in the top coordinate system.

$\mathbf{r}$ = position of the origin of the top coordinate system in the fixed coordinate system.

The vectors along the limbs in the fixed coordinate system are given by:

$$\mathbf{l_i} = \mathbf{s_i} - \mathbf{p_i}$$

The limb lengths are the magnitude of this vector:

$$l_i = \frac{\mathbf{l_i}}{|\mathbf{l_i}|}$$

For each limb, a coordinate system attached to the limb is setup. The limbs will be assumed to be axisymmetric. The z-axis for the new limb coordinate system is along the limb. The x-axis is perpendicular to the z-axis of the limb coordinate system and the x-axis of the fixed coordinate system. Thus the rotation matrix is given by:

$$ {}^{b}\mathbf{R}_{l_i} = \begin{bmatrix} \sqrt{1-l_{ix}^2} & 0 & l_{ix} \\ -\dfrac{l_{ix}l_{iy}}{\sqrt{1-l_{ix}^2}} & \dfrac{l_{iz}}{\sqrt{1-l_{ix}^2}} & l_{iy} \\ -\dfrac{l_{ix}l_{iz}}{\sqrt{1-l_{ix}^2}} & -\dfrac{l_{iy}}{\sqrt{1-l_{ix}^2}} & l_{iz} \end{bmatrix} $$

where

l_{ix}, l_{iy}, l_{iz} = components of the vector $\mathbf{l_i}$.

The velocity of the ith joint is given by:

$$ {}^{t}\dot{\mathbf{s}}_i = {}^{t}\mathbf{v} + {}^{t}\boldsymbol{\omega} \times {}^{t}\mathbf{q_i} $$

where

${}^{t}\mathbf{v}$ = velocity of the center of mass (coincident with the origin) of the top member.

${}^{t}\boldsymbol{\omega}$ = angular velocity of the center of mass of the top member.

The velocity of the ith joint in the limb coordinate system is:

$$ {}^{l_i}\dot{\mathbf{s}}_i = {}^{l_i}\mathbf{R}_t \, {}^{t}\dot{\mathbf{s}}_i $$

where

$ {}^{l_i}\mathbf{R}_t = {}^{l_i}\mathbf{R}_b \, {}^{b}\mathbf{R}_t $

In the following section, all the variables are in the limb coordinate system. The superscript is avoided for clarity.

The angular velocity of the ith limb in the limb coordinate system is given by:

$$\omega_{ix} = -\frac{\dot{s}_{iy}}{l}$$

$$\omega_{iy} = -\frac{\dot{s}_{ix}}{l}$$

The velocity of the prismatic joint is given by:

$$\dot{l} = \dot{s}_{iz}$$

The required equations of motion of the bottom link are:

$$T_{i2x} = \left(J_{i1x} + m_{i1}b_{i11}^2\right)\alpha_{ix} + b_{i1}F_{i2y} + b_{i11}m_{i1}g_{iy}$$

$$T_{i2y} = \left(J_{i1y} + m_{i1}b_{i11}^2\right)\alpha_{iy} + b_{i1}F_{i2x} + b_{i11}m_{i1}g_{ix} \quad (a1)$$

where

J_{i1x} = mass moment of inertia of the lower limb about the x-axis.

J_{i1y} = mass moment of inertia of the lower limb about the y-axis.

$\begin{bmatrix} g_{ix} \\ g_{iy} \\ g_{iz} \end{bmatrix} = {}^{l_i}\mathbf{R}_b \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$ = gravity vector in the limb coordinate system.

g = acceleration due to gravity (9.81 m/s^2).

The equation of motion of the top limb are given by:

$$m_{i2}\left[(l_i - b_{i22})\alpha_{iy} + 2\dot{l}_i\omega_{iy}\right] = F_{i3x} - F_{i2x} + m_{i2}g_{ix}$$

$$m_{i2}\left[-(l_i - b_{i22})\alpha_{ix} - 2\dot{l}_i\omega_{ix}\right] = F_{i3y} - F_{i2y}$$

$$+m_{i2}g_{iy}$$

$$m_{i2}\left[\ddot{l}_i - (l_i - b_{i22})\left(\omega_{ix}^2 + \omega_{iy}^2\right)\right] = F_{i3z} - F_{i2z}$$

$$+m_{i2}g_{iz}$$

$$J_{i2x}\alpha_{ix} = -b_{i22}F_{i3y} - b_{i21}F_{i2y} - T_{i2x}$$

$$J_{i2y}\alpha_{iy} = b_{i22}F_{i3x} + b_{i21}F_{i2x} - T_{i2y}$$

$$J_{i2z}\alpha_{iz} = T_{i3x} \quad (a2)$$

where

$\alpha_{ix}, \alpha_{iy}, \alpha_{iz}$ = angular acceleration of the limb in the limb coordinate system.

m_{i2} = mass of the upper part of ith limb.

$J_{i2x}, J_{i2y}, J_{i2z}$ = mass moment of inertia of the upper part of ith limb about its center of mass.

Combining equations (a1) and (a2):

$$\alpha_{iy} = c_{i11}F_{i3x} + c_{i12}$$

$$\alpha_{ix} = c_{i21}F_{i3y} + c_{i22}$$

$$\ddot{l}_i = c_{i31}F_{i3x} + c_{i32} \quad (a3)$$

where

$$c_{i11} = \frac{l_i}{J_{i1y} + J_{i2y} + m_{i1}b_{i11}^2 + m_{i2}(l_i - b_{i22})^2}$$

$$c_{i21} = -\frac{l_i}{J_{i1x} + J_{i2x} + m_{i1}b_{i11}^2 + m_{i2}(l_i - b_{i22})^2}$$

$$c_{i12} = \frac{\left[m_{i1}b_{i11} + m_{i2}(l_i - b_{i22})\right]g_{ix}}{J_{i1y} + J_{i2y} + m_{i1}b_{i11}^2 + m_{i2}(l_i - b_{i22})^2}$$

$$-\frac{2m_{i2}\dot{l}_i\omega_{iy}(l_i - b_{i22})}{J_{i1y} + J_{i2y} + m_{i1}b_{i11}^2 + m_{i2}(l_i - b_{i22})^2}$$

$$c_{i22} = -\frac{\left[m_{i1}b_{i11} + m_{i2}(l_i - b_{i22})\right]g_{iy}}{J_{i1x} + J_{i2x} + m_{i1}b_{i11}^2 + m_{i2}(l_i - b_{i22})^2}$$

$$-\frac{2m_{i2}\dot{l}_i\omega_{ix}(l_i - b_{i22})}{J_{i1x} + J_{i2x} + m_{i1}b_{i11}^2 + m_{i2}(l_i - b_{i22})^2}$$

$$c_{i31} = \frac{1}{m_{i2}}$$

$$c_{i32} = \frac{F_{ia} + m_{i2}g_{iz} + m_{i2}(l_i - b_{i22})\left(\omega_{ix}^2 + \omega_{iy}^2\right)}{m_{i2}}$$

F_{ia} = force from the actuator

The angular acceleration in equation (a3) are related to the acceleration of the top joint by the equation:

$$\alpha_{ix} = \frac{-a_{i3y} - 2\dot{l}_i\omega_{ix}}{l_i}$$

$$\alpha_{iy} = \frac{a_{i3x} - 2\dot{l}_i\omega_{iy}}{l_i} \quad (a4)$$

Combining equations (a3) and (a4):

$$\mathbf{a}_{i3} = \mathbf{D}_i\mathbf{F}_{i3} + \mathbf{d}_i \quad (a5)$$

where

$$\mathbf{a}_{i3} = \begin{bmatrix} a_{i3x} & a_{i3y} & a_{i3z} \end{bmatrix}^T$$

$$\mathbf{F}_{i3} = \begin{bmatrix} F_{i3x} & F_{i3y} & F_{i3z} \end{bmatrix}^T$$

$$\mathbf{D}_i = \begin{bmatrix} c_{i11}l_i & 0 & 0 \\ 0 & -c_{i21}l_i & 0 \\ 0 & 0 & c_{i31} \end{bmatrix}$$

$$\mathbf{d}_i = \begin{bmatrix} c_{i12} l_i + 2\dot{i}\omega_{iy} \\ -c_{i22} l_i - 2\dot{i}\omega_{ix} \\ c_{i32} - l\left(\omega_{ix}^2 + \omega_{iy}^2\right) \end{bmatrix}$$

This matrices $\mathbf{D}_i$ and $\mathbf{d}_i$ in this equation can be computed for each limb.

Appendix B

The model for the two stage servovalves for small displacements is given by the block diagram in Fig. 1.

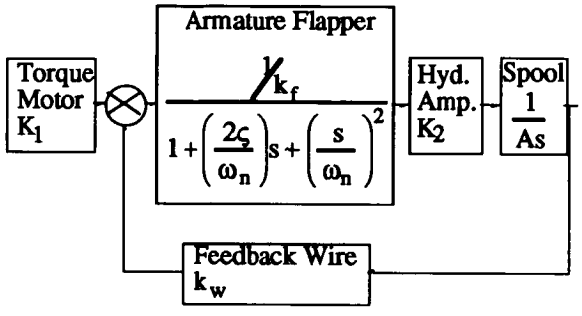


Figure b1. Block Diagram of Flow Control hydraulic servovalve ([8]).

The flow through an orifice is given by the equation:

$$Q = KA\sqrt{p_1 - p_2} \tag{b1}$$

where
Q = volume flow rate.
A = area of the orifice.
p_1 = pressure on one side of the orifice.
p_2 = pressure on the other side of the orifice.
K = constant.

For a positive displacement of the servovalve x_S, the supply pressure is connected to chamber 2 of the cylinder and the return side is connected to the chamber 1. The equations are given by:

$$Q_{in} = K_s x_s \sqrt{p_s - p_2} + K_l(p_1 - p_2) \tag{b2}$$

$$Q_{out} = K_s x_s \sqrt{p_1 - p_r} - K_l(p_1 - p_2) \tag{b3}$$

These flow rates are equal to the corresponding volume changes in the chambers. These change are given by:

$$Q_{in} = \dot{i}A_2 + \frac{A_2(1 - b_2)}{\beta}\frac{dp_2}{dt} \tag{b4}$$

$$Q_{out} = \dot{i}A_1 - \frac{A_1(b_1 + b_2 - 1)}{\beta}\frac{dp_1}{dt} \tag{b5}$$

Using the above set of equations:

$$\frac{A_2(1 - b_2)}{\beta}\frac{dp_2}{dt} = K_s|x_s|\sqrt{p_s - p_2} - \dot{i}A_2 + K_l(p_1 - p_2) \tag{b6}$$

$$\frac{A_1(b_1 + b_2 - 1)}{\beta}\frac{dp_1}{dt} = -K_s|x_s|\sqrt{p_1 - p_r} + \dot{i}A_1 - K_l(p_1 - p_2) \tag{b7}$$

If x_S is negative, then these equations are:

$$\frac{A_2(1 - b_2)}{\beta}\frac{dp_2}{dt} = -K_s|x_s|\sqrt{p_2 - p_r} - \dot{i}A_2 + K_l(p_1 - p_2) \tag{b8}$$

$$\frac{A_1(b_1 + b_2 - 1)}{\beta}\frac{dp_1}{dt} = K_s|x_s|\sqrt{p_s - p_1} + \dot{i}A_1 - K_l(p_1 - p_2) \tag{b9}$$

The above set of differential equations can be solved to obtain the pressure in the cylinder chambers. The force on the limb is given by:

$$F_{ia} = p_2 A_2 - p_1 A_1 - c\dot{i} \tag{b10}$$

where
c = damping coefficient.

This completes the development of the equations for the hydraulic cylinder.

AIAA-94-1299-CP

ROBUST INVERSE KINEMATICS USING DAMPED LEAST SQUARES WITH DYNAMIC WEIGHTING

D. E. Schinstock, T. N. Faddis, R. B. Greenway
University of Kansas, Mechanical Engineering
Lawrence Kansas

Abstract

A method of dealing with singularities and joint limits in the inverse kinematics for both redundant and non-redundant serial-link manipulators is presented. The method uses damped least squares with dynamic weighting for the approximate solution of the inverse Jacobian problem. Damped least squares has become a popular approach for dealing with singularities. The method presented extends the utility of damped least squares by incorporating dynamic weighting matrices within its formulation. This allows specific joints to be targeted in the minimization of the joint differential vector. An efficient algorithm is given for the solution of the weighted damped least squares problem. This algorithm is implemented, along with an algorithm to set the weights, for a six d.o.f. telemanipulator slave. A solution that is approximate in the task space and that is physically realizable in the joint space is obtained at or near singularities and/or joint limits. Away from singularities and joint limits an exact solution is obtained. The results are a well behaved slave manipulator under teleoperational control even when the slave is at or near singularities and/or when unreachable configurations are commanded.

1 Introduction

The inverse kinematics problem can be stated as follows: given a desired position and orientation of the end effector of a manipulator find a joint configuration that satisfies it. This problem is central in the control of robot manipulators. Any time the motions of a manipulator are described in a general space such as a Cartesian space, the inverse kinematics must be solved. In order to avoid this, manipulators are often controlled by describing the motions only in the joint space. This is done, however, with a great loss of generality. For serial-link manipulators the inverse kinematics problem is complicated by non-linearities, singularities, unreachable configurations, multiple solutions, and even infinite solutions in the case of a redundant manipulator. The nonlinearities can be avoided by calculating the inverse kinematics iteratively using the Jacobian of the manipulator. Redundancies, while complicating the solutions, are actually utilized to satisfy some criteria that is secondary to the motion of the end effector. This is a large body of research. The focus of this paper is in dealing with singularities and unreachable configurations.

The methods discussed here utilize the damped least squares inverse of the Jacobian, which has been proposed by many researchers for the inverse kinematics problem.[1,2,3,4,5] This inverse has the benefit of being compatible with solutions based on the pseudoinverse which has become a very popular method of calculating the inverse Jacobian. The pseudoinverse has become popular for many reasons including the utilization of redundancy.[6,7,8,9] The use of damped least squares results in an approximate solution with a decrease in the size of the solution vector. This is beneficial in controlling the large joint rates resulting from exact solutions near singularities. The addition of dynamic weighting matrices in the damped least squares solution is proposed in this paper, to increase its utility. Using weighting matrices the damped least squares solution is extended to methods of dealing with unreachable configurations caused by joint limits. The previous research with damped least squares has dealt mainly with singularities. The dynamic weights can also be used to target the problem joints, of a particular singularity in reducing the size of the joint space solution vector.

For any serial-link manipulator a particular configuration of the joints corresponds to a unique position and orientation of the end effector in Cartesian space. This relationship is described by the forward kinematics function of the manipulator. The methods used to develop this function are well established. The position and orientation, or the task space variable, $\mathbf{x} \in R^m$ (generally m=6), of the end effector is described as a function of the joint space variable, $\mathbf{q} \in R^n$, by the nonlinear forward kinematics equation,

$$\mathbf{x} = \Lambda(\mathbf{q}) \tag{1}$$

Copyright © 1994 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved

The differential relationship of $\mathbf{x}$ and $\mathbf{q}$ is described using the following linear equation:

$$\delta \mathbf{x} = \mathbf{J}(\mathbf{q})\delta \mathbf{q} \quad \text{where} \quad \mathbf{J}(\mathbf{q}) \equiv \frac{\partial \mathbf{x}/\partial t}{\partial \mathbf{q}/\partial t} \qquad (2)$$

In equation (2) $\mathbf{J}(\mathbf{q})$ is the $m \times n$ manipultor Jacobian matrix. General methods for the development of the forward kinematics function and the Jacobian of a manipulator can be found in any introductory text on robotics such as Craig[10], Paul[11], or Koivo[12]. (Note: Hereafter the functional dependence of $\mathbf{J}$ on $\mathbf{q}$ will be dropped and assumed to be understood.)

The inverse kinematics problem almost always reduces to that of solving equation (1) for $\mathbf{q}$ or equation (2) for $\delta\mathbf{q}$ in an iterative scheme to find $\mathbf{q}$. Analytical solutions of equation (1) are known only for a few simple non-redundant (m=n) manipulator geometries. The six degree of freedom (m=n=6) manipulator geometries for which these solutions exist were clarified by Pieper[13]. Iterative inverse kinematics schemes can be used by solving equation (2). This can be done off line or it can be done on line within the control system of the manipulator, using each iteration to calculate the joint control law. An example of on-line iterative inverse kinematics is resolved rate control.[14] Nearly all of the contemporary research dealing with inverse kinematics solutions is devoted to finding a solution or an approximate solution to equation (2). This is true for three reasons: 1) there are many manipulators for which an analytical solution to (1) does not exist; 2) the nonlinearities of equation (1) impede the development of general methods for a numerical solution procedure; 3) general methods for dealing with the other complications of inverse kinematics can be incorporated in the solution of (2). Solutions to equation (2) are commonly found by solving a linear system of equations for $\delta\mathbf{q}$ using some well established method, such as Gaussian elimination. In the non-redundant, exact case these solutions are described using the equation

$$\delta \mathbf{q} = \mathbf{J}^{-1}\delta \mathbf{x} \qquad (3)$$

This equation may be generalized to include redundant manipulators and/or approximate solutions using the following equation:

$$\delta \mathbf{q} = \mathbf{J}^{\#}\delta \mathbf{x} \qquad (4)$$

In this equation $\mathbf{J}^{\#}$ is a some type of inverse of the Jacobian matrix. If m=n then $\mathbf{J}^{\#}$ is likely to be $\mathbf{J}^{-1}$, whereas for redundant manipulators (n>m) $\mathbf{J}^{\#}$ might be the Moore-Penrose pseudoinverse[15], $\mathbf{J}^{*} \equiv \mathbf{J}^{T}(\mathbf{J}\mathbf{J}^{T})^{-1}$. In the redundant case with the pseudoinverse, (4) is a particular solution, the minimum norm solution, of the general solution given by

$$\delta \mathbf{q} = \mathbf{J}^{\#}\delta \mathbf{x} + (\mathbf{I} - \mathbf{J}^{\#}\mathbf{J})\,\mathbf{v} \qquad (5)$$

Here, $\mathbf{v} \in R^{n}$ is an arbitrary joint space vector used to satisfy some criterion such as obstacle avoidance. The null space vector , v, is projected into the null space of the Jacobian, taking advantage of the redundancy. Therefore, the solution given by (5) still satisfies $\delta \mathbf{x} = \mathbf{J}\delta \mathbf{q}$. It might be noted that in the non-redundant case where $\mathbf{J}^{\#}=\mathbf{J}^{-1}$, the second term of equation (5) vanishes and the null space vector has no effect.

2 Singularities and Workspace Boundaries

The solutions to the inverse kinematics problem given in the previous section work well for control of a manipulator when it is not near a singularity or workspace boundary. However, near a singularity or workspace boundary certain components of commanded movements either require large joint rates or are physically impossible to satisfy. Therefore a robust algorithm for the calculation of the inverse kinematics of a manipulator must deal with singularities and workspace boundaries.

Physically, a singularity may be described using end effector motions (or forces) in the task space. In a singular configuration a manipulator is degenerate, causing the end effector to loose degrees of freedom in the task space. This means that the robot cannot move (control forces) in certain directions or that motion (force) in some direction is dependent upon motion (force) in others. Near a singularity small motions (large forces) in certain directions require large joint rates (small joint torques).

Mathematically a singularity may be described using the Jacobian matrix. The Jacobian is rank deficient (rank(J)<m) when the manipulator is in a singular configuration. In the case of m=n the determinate of the Jacobian is zero. Near a singularity the Jacobian becomes ill-conditioned and elements of the inverse or pseudoinverse are large. If the condition number of the

Jacobian becomes too large then a general solution attempting to solve equation (2) will have numerical problems.

A significant amount of research is devoted to developing inverse kinematics with singularity robustness. Methods that utilize redundancy to avoid singularities have been proposed by many researchers.[16,17,18,19] However none of these methods ensure both a nonsingular Jacobian and non-cyclic behavior. Also, they need the null space vector which might be used for other purposes. Whitney[14] proposed removing the under generating block of the Jacobian and then using a pseudoinverse to calculate an approximate solution near singularities. However this requires a different Jacobian for each singularity. Futhermore, achieving continuity when switching solutions is difficult with this method. The most appealing of the proposed solutions are those that use damped least squares.[1,2,3,4,5]

Workspace boundaries are the boundaries between that space which is reachable by the manipulator ($W \subset R^m$) and that space which is not ($\overline{W}$). These boundaries occur in configurations where the manipulator is at a singularity(s) or at a joint limit(s). Because of joint limits an algorithm that deals with all of the singularities of a manipulator does not necessarily deal with all of the workspace boundaries.

Few solutions to the joint limit problem have been given in the literature. Some propose the use of redundancy to avoid joint limits.[6] The methods proposed do not ensure their avoidance and are not applicable in cases where redundancy is not available. In practice this problem has been dealt with at the global level of path planning, searching the entire path for unreachable configurations. However, this is not always possible such as when the manipulator is operated teleoperationally with a human giving real time commands.

3 Inverse Kinematics Using Damped Least Squares

The damped least squares solutions to the inverse kinematics problem are intended to ensure a well conditioned matrix for the solution algorithm while limiting the size of the solution vector, $\delta\mathbf{q}$. This is done by adding a diagonal matrix, $\alpha\mathbf{I}$, to the matrix $\mathbf{J}\mathbf{J}^T$. Damped least squares may be used when an approximate solution to equation (2) is necessary or acceptable.

If $\delta\mathbf{q}$ is found using the equation $\delta\mathbf{q} = \mathbf{J}^*\delta\mathbf{x}$, where $\mathbf{J}^*$ is the pseudoinverse. Then the solution, $\delta\mathbf{q}$, satisfies

$$\min_{\delta\mathbf{q}} \|\delta\mathbf{q}\|^2 \tag{6}$$

among all $\delta\mathbf{q}$ satisfying

$$\min_{\delta\mathbf{q}} \|\delta\mathbf{x} - \mathbf{J}\delta\mathbf{q}\|^2 \tag{7}$$

where $\|\cdot\|$ denotes the Euclidean norm. If the Jacobian is of full rank then satisfying the constraint (7) is the same as satisfying equation (2).

If the approximate solution of damped least squares, $\delta\mathbf{q} = \mathbf{J}^+\delta\mathbf{x}$ where

$$\mathbf{J}^+ = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \alpha\mathbf{I})^{-1}, \quad \alpha \geq 0, \tag{8}$$

is used then the solution satisfies

$$\min_{\delta\mathbf{q}} \left\{ \|\delta\mathbf{x} - \mathbf{J}\delta\mathbf{q}\|^2 + \alpha^2 \|\delta\mathbf{q}\|^2 \right\} \tag{9}$$

Note that for α=0 the damped least squares solution is the pseudoinverse solution.

In the damped least squares case the size of error in the task space is weighed against the size of the resulting solution. For a given $\delta\mathbf{x}$ the size of the solution vector is decreased by increasing α. However, this is done at the expense of using an approximate solution. As α increases so does the size of the error in the task space. A large α has the other benefit of ensuring a well conditioned matrix for inversion. It has been shown by Mayorga et al.[1] that the condition number, κ, of the matrix $\mathbf{P} \equiv (\mathbf{J}\mathbf{J}^T + \alpha\mathbf{I})$, is

$$\kappa = (\sigma_1^2 + \alpha)/(\sigma_m^2 + \alpha) \tag{10}$$

where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m \geq 0$ are the singular values of the Jacobian matrix. It can be seen in this equation that κ is made arbitrarily close to 1 by increasing α, thus ensuring a well conditioned matrix for inversion even in a singular configuration where σ_m=0.

Simply stated, if one is willing to give up the exactness of the solution then the size of the joint rates

can be reduced and a well conditioned matrix for inversion can be ensured by increasing α. It should be pointed out that an exact solution may not be physically obtainable or even desirable near a singularity. This is due to the loss of degrees of freedom in the task space or the necessity of large joint rates, which may be unachievable or beyond safety limits.

4 The Addition of Weighting to Damped Least Squares

The weighted damped least squares solution is intended to increase the utility of the damped least squares solution by providing the ability to significantly affect the size of individual components of mimimized vectors. This is done by adding weighting matrices to the formulation of the damped least squares problem.

The importance of individual components in the minimization condition of the damped least squares solution can be adjusted by scaling the rows and columns of the Jacobian before the damped least squares solution is calculated. Consider the following reformulation of equation (2):

$$\mathbf{W}_x \delta \mathbf{x} = \mathbf{W}_x \mathbf{J} \mathbf{W}_q \mathbf{W}_q^{-1} \delta \mathbf{q} \tag{11}$$

where the weighting matrices are defined by

$$\begin{aligned} \mathbf{W}_x &\equiv \mathrm{diag}[w_{x_1} \ldots w_{x_m}], \ w_{x_i} > 0 \\ \mathbf{W}_q &\equiv \mathrm{diag}[w_{q_1} \ldots w_{q_n}], \ w_{q_i} > 0 \end{aligned} \tag{12}$$

If the following definitions are made

$$\begin{aligned} &\delta \mathbf{x}_w \equiv \mathbf{W}_x \delta \mathbf{x}, \ \delta \mathbf{q}_w \equiv \mathbf{W}_q^{-1} \delta \mathbf{q}, \\ &\mathbf{J}_w \equiv \mathbf{W}_x \mathbf{J} \mathbf{W}_q, \ \mathbf{P}_w \equiv \mathbf{J}_w \mathbf{J}_w^T + \alpha \mathbf{I} \end{aligned} \tag{13}$$

then (11) can be written

$$\delta \mathbf{x}_w = \mathbf{J}_w \delta \mathbf{q}_w \tag{14}$$

Solving (14) using damped least squares $(\delta \mathbf{q}_w = \mathbf{J}_w^T (\mathbf{J}_w \mathbf{J}_w^T + \alpha \mathbf{I})^{-1} \delta \mathbf{x}_w = \mathbf{J}_w^+ \delta \mathbf{x}_w)$ results in an approximate solution to equation (2), given by

$$\delta \mathbf{q} = \mathbf{W}_q \delta \mathbf{q}_w = \mathbf{W}_q \mathbf{J}_w^+ \delta \mathbf{x}_w \tag{15}$$

This solution satisfies

$$\min_{\delta \mathbf{q}} \left\{ \left\| \mathbf{W}_x (\delta \mathbf{x} - \mathbf{J} \delta \mathbf{q}) \right\|^2 + \alpha^2 \left\| \mathbf{W}_q^{-1} \delta \mathbf{q} \right\|^2 \right\} \tag{16}$$

Taking the diagonal structure of the weighting matrices into account, it can be seen in the minimization condition (16) that the relative importance of the individual components of the task space error vector and of the solution vector are controlled by the individual elements of $\mathbf{W}_x$ and $\mathbf{W}_q$ respectively. If $\alpha > 0$ then increasing the size of w_{x_i} decreases the size of $(\delta \mathbf{x} - \mathbf{J} \delta \mathbf{q})_i$ and decreasing the size of w_{q_i} decreases the size of $\delta \mathbf{q}_i$. The strict inequalities in (12) may be relaxed for $\alpha > 0$ allowing $w_{q_i} = 0$ for some i. This may be desired if it is necessary to eliminate the use of some $\delta \mathbf{q}_i$ from the solution, such as at a joint limit.

5 Solution Algorithm

In solving the inverse kinematics it is desirable to avoid the explicit inversion of a matrix since this is computationally expensive. A more efficient algorithm will solve a linear system of equations using Guassian elimination or some similar method. It is also desirable to minimize the number of matrix-matrix multiplies and to factor the matrix-matrix and matrix-vector multiplication. The following reformulation of the problem, will aid in the understanding of the solution algorithm presented here.

$$\begin{aligned} \delta \mathbf{q} &= \mathbf{W}_q \mathbf{J}_w^+ \delta \mathbf{x}_w \\ &= \mathbf{W}_q \mathbf{J}_w^T (\mathbf{J}_w \mathbf{J}_w^T + \alpha \mathbf{I})^{-1} \delta \mathbf{x}_w \\ &= \mathbf{W}_q \mathbf{J}_w^T \mathbf{P}_w^{-1} \delta \mathbf{x}_w \\ &= \mathbf{W}_q \mathbf{J}_w^T \mathbf{y} \end{aligned} \tag{17}$$

where $\mathbf{y} = \mathbf{P}_w^{-1} \delta \mathbf{x}_w$.

An efficient solution algorithm for inverse kinematics problem using weighted damped least squares is summarized in the following five steps:

1. calculate or set $\mathbf{J}, \delta \mathbf{x}, \mathbf{W}_x, \mathbf{W}_q$, and α
2. form $\delta \mathbf{x}_w = \mathbf{W}_x \delta \mathbf{x}$ and $\mathbf{J}_w = \mathbf{W}_x \mathbf{J} \mathbf{W}_q$
3. form $\mathbf{P}_w = \mathbf{J}_w \mathbf{J}_w^T + \alpha \mathbf{I}$
4. solve $\delta \mathbf{x}_w = \mathbf{P}_w \mathbf{y}$ for y using Cholesky decompostion
5. form $\delta \mathbf{q} = \mathbf{W}_q \mathbf{J}_w^T \mathbf{y}$

It is not claimed that this is the computationally optimal algorithm for solving this problem. An optimal algorithm is both application and hardware dependent. However, this algorithm is fairly efficient if the implementation takes advantage of the structures of the matrices and vectors involved. Most important are the diagonal structure of $\mathbf{W}_x$ and $\mathbf{W}_q$, and the symmetry of $\mathbf{P}_w$. Additionally, $\mathbf{P}_w$ is positive definite if: 1) $\text{rank}(\mathbf{J}_w) = m$ and $\alpha \geq 0$, or 2) $\alpha > 0$. $\mathbf{J}_w$ can only be rank deficient if $\mathbf{J}$ is rank deficient or some weight is set to zero. In either of these cases, α will be significantly larger than zero if it is set correctly. Therefore, Cholesky decomposition can be used for the solution of the linear system in step 4. One might note that the two conditions given above, are actually the conditions that ensure $\mathbf{P}_w$ is nonsingular, which is important in any solution scheme.

The algorithm above does not include the utilization of a null space vector for self motion of redundant manipulators. A algorithm similar to the one above which utilizes the a null space vector requires a reformulation of the general solution to equation (2). Such a reformulation follows.

$$\begin{aligned}
\delta\mathbf{q} &= \mathbf{W}_q\{\mathbf{J}_w^+\delta\mathbf{x}_w + (\mathbf{I}-\mathbf{J}_w^+\mathbf{J}_w)\mathbf{v}_w\} \\
&= \mathbf{W}_q\{\mathbf{J}_w^T\mathbf{P}_w^{-1}\delta\mathbf{x}_w + (\mathbf{I}-\mathbf{J}_w^T\mathbf{P}_w^{-1}\mathbf{W}_x\mathbf{J}\mathbf{W}_q)\mathbf{W}_q^{-1}\mathbf{v}\} \\
&= \mathbf{W}_q\{\mathbf{J}_w^T\mathbf{P}_w^{-1}(\delta\mathbf{x}_w - \mathbf{W}_x\mathbf{J}\mathbf{v}) + \mathbf{W}_q^{-1}\mathbf{v}\} \qquad (18) \\
&= \mathbf{W}_q\mathbf{J}_w^T\mathbf{P}_w^{-1}\mathbf{W}_x(\delta\mathbf{x}-\mathbf{J}\mathbf{v}) + \mathbf{v} \\
&= \mathbf{W}_q\mathbf{J}_w^T\mathbf{y} + \mathbf{v}
\end{aligned}$$

where $\mathbf{y} = \mathbf{P}_w^{-1}\mathbf{W}_x(\delta\mathbf{x}-\mathbf{J}\mathbf{v})$.

An algorithm similar to the one above which implements the general solution is summarized by the following five steps.

1. calculate or set $\mathbf{J}, \delta\mathbf{x}, \mathbf{v}, \mathbf{W}_x, \mathbf{W}_q$, and α
2. form $\mathbf{z} = \mathbf{W}_x(\delta\mathbf{x}-\mathbf{J}\mathbf{v})$ and $\mathbf{J}_w = \mathbf{W}_x\mathbf{J}\mathbf{W}_q$
3. form $\mathbf{P}_w = \mathbf{J}_w\mathbf{J}_w^T + \alpha\mathbf{I}$
4. solve $\mathbf{z} = \mathbf{P}_w\mathbf{y}$ for y using Cholesky decomposition
5. form $\delta\mathbf{q} = \mathbf{W}_q\mathbf{J}_w^T\mathbf{y} + \mathbf{v}$

6 Setting the Weighting Matrices and the Damping Factor

6.1 General Discussion of the Weighting Matrices and the Damping Factor

Developing an algorithm to set the weights and damping factor is not a trivial task. There are several requirements of this algorithm. The basic idea is to dynamically adjust the weights and damping factor so that: 1) an exact solution is obtained away from singularities and joint limits; 2) an approximate solution, that is near the desired solution in the task space and is physically realizable in the joint space, is obtained near singularities and joint limits; 3) the transitions between exact and approximate solutions and between different approximate solutions are smooth.

A non-zero damping factor, α, is used to ensure a well conditioned matrix and to include the solution vector in the minimized quantity. However, for $\alpha>0$ the solution is an approximate one. Therefore, away from singularities and joint limits, α should be zero. As the manipulator approaches a singularity or joint limit α should be increased in a manner that: 1) ensures a stable numerical solution; 2) keeps the joint differentials within safe limits. A value of α that satisfies the second condition will most probably satisfy the first. It should be noted that the first condition is only a concern near singularities or when there is a very large relative difference in the size of the individual weights that are used. It should also be noted that satisfying the second condition is dependent upon the method used to set the joint weights.

The relative sizes of the joint weights are used to control the importance of each of the joints in the minimization. If a joint has a small weight and α is non-zero, then the approximate solution tends not to use that joint. For example, in a region near a singularity where the rate for joint i tends to be large, w_{q_i} should be small. The weight might also be small near a limit for joint i. However, setting a joint weight based solely on the distance from the joint limit poses a problem in itself. If this is done, the joint will tend to stay at the limit even if the exact solution gives a δq_i which is away from the limit. Therefore it is necessary to include another criterion such as the direction of δq_i in the previous iteration of the inverse kinematics.

The relative sizes of the error vector weights can be used to control the importance of individual components of the task space error vector. Increasing w_{x_i} decreases the size of $(\delta \mathbf{x} - \mathbf{J}\delta\mathbf{q})_i$. While there are certainly situations in which it would be useful to dynamically set $\mathbf{W_x}$, what is suggested here is use of a constant $\mathbf{W_x}$ to normalize the Jacobian matrix. For m=6 the Jacobian contains three rows related to position and three rows related to orientation. The units in which position and orientation are measured create large differences in the relative magnitude of the elements of $\delta \mathbf{x}$. A constant $\mathbf{W_x}$ can be used to balance the importance of position and orientation.

6.2 Previous Work Dealing With Setting the Damping Factor

Several schemes have been proposed for the computation of the damping factor in damped least squares without dynamic weighting. Most of these schemes are aimed at providing singularity robustness and do not include provisions for joint limits. Nakamura and Hanafusa[2] proposed that the damping factor be calculated as follows:

$$\alpha = \begin{cases} \alpha_0 (1 - w/w_0)^2 & \text{if } w < w_0 \\ 0 & \text{if } w \geq w_0 \end{cases} \qquad (19)$$

where $w = \sqrt{\det(\mathbf{J}\mathbf{J}^T)}$ is Yoshikawa's manipulability measure[20] and α_0 and w_0 are constants to be determined experimentally. A modification to this scheme is given Kelmar and Kholsa[3]. They suggest a method of calculating the damping factor using the manipulability measures at the ith and (i+1)th iterations. Mayorga et al.[1] proposed a scheme which establishes an upper bound, ℓ_0, for the condition number of the matrix $(\mathbf{J}\mathbf{J}^T + \alpha\mathbf{I})$. In their scheme α is calculated for the next iteration using parameters calculated in the present iteration and the three constants α_0, ℓ_0, and m. The method is described as follows:

$$\alpha_{i+1} = \begin{cases} \alpha_0 (1 - \ell/\ell_0)^2 & \text{if } \ell > \ell_0 \\ 0 & \text{if } \ell \leq \ell_0 \end{cases} \qquad (20)$$

where $\ell = m\zeta^4 d_g / d_s$. Here ζ is the upper bound of the infinity norm $\|\mathbf{L}\|_\infty$, d_g and $d_s$are the greatest and smallest elements, respectively, of the diagonal matrix D, and L is the unit triangular matrix such that $(\mathbf{J}\mathbf{J}^T + \alpha\mathbf{I}) = \mathbf{L}\mathbf{D}\mathbf{L}^T$. It is shown that the parameters ζ, d_g, and d_s can be calculated inexpensively as by-products of Guassian elimination in the solution of the damped least squares problem. A method was suggested by Chan and Lawrence[4] for the calculation of the damping factor for both singularity and workspace boundary robustness. They give the equation

$$\alpha = \alpha_0 \|\delta \mathbf{x}_e\|^2 \qquad (21)$$

where $\delta \mathbf{x}_e$ is the error of the manipulator in the task space and α_0 is a constant. None of the methods given in first three references[1,2,3] provide for a non-zero damping factor near joint limits. This is because they are established only for singularity robustness. The last scheme[4], which does include provisions for joint limits, has questionable performance in providing a well conditioned matrix near singularities. Also, its performance with load generated task space errors is questionable. All of these methods are intended to be used throughout the workspace of the manipulator. While it is theoretically justified to find a general method to handle all cases, perhaps a more effective solution would be to find measures which target specific singularities and joint limits. Such measures could also be used in the setting of the joint weights to specifically target the problem joint(s) for that singularity.

6.3 Scheme for Setting the Weighting Matrices and the Damping Factor

A scheme for setting the damping factor and joint weights is presented here. It considers both singularities and joint limits. The equations for this scheme, (22) through (29), are given below. In each iteration of the kinematic calculations, it determines a damping factor and a set of joint weights for each singularity. It also determines a damping factor and a joint weight for each pair of joint limits. However, only the maximum damping factor, and the minimum weight for each joint, are used in the inverse kinematics solution (equations (28) and (29)).

The damping factor ranges in value from zero, when the manipulator is not in the region of a singularity or limit, to α_0, when the manipulator is at a singularity or limit. The joint weights range in value from one, away from singularities and limits, to w_{q0s} at a singularity, and to w_{q0l} at a joint limit. Different minimum values for the joint weights are used at singularities and limits

because at a limit it is desired to completely eliminate the joint from the solution, while at singularity it might only be desired to control the rate of the joint. To prevent sticking at the limit, zero is not used. In practice a small joint differential obtained using a small joint weight, rather than zero, can be discarded.

For the singularities the damping factor and joint weights are set using a measure of distance from the singularity (equations (22) and (23)). A measure of distance, either a linear distance or an angle, must be identified for each singularity that is reachable by the manipulator. The joints that need additional damping also need to be identified for each singularity. If it is not possible to make these identifications, then (22) might be replaced by a scheme similar to those given in (19) or (20). However, the benefits of dynamic joint weighting are lost, for singularities, if these schemes are used.

Because a joint limit is a one sided problem, temporal ramps are used in setting the damping factor and weights, when considering joint limits (equations (26) and (27)). These ramps are calculated using: one as an upper bound; a dynamic lower bound found using the distance from the limit the joint is moving towards; and a discrete step (equations (24) and (25)). If a joint is in a joint limit region and moving towards the limit, then its weight is ramped down to a value that is found using the distance from limit. If it is moving away from the limit, then its weight is ramped back up to 1.

The following nomenclature and equations are defined for the algorithm used to set the damping factor and joint weights.

Constants

α_0 ..... upper limit of damping factor

$w_{q0s}, w_{q0\ell}$ ..... lower limits of joint weights for singularities and joint limits

$d_{0s_j}, d_{0\ell}$ ..... edge the region for singularity j and edge of region for joint limits

δu ..... step to increase or decrease the joint limit ramps in successive iterations

$hilim_i, lolim_i$ .... high and low limit for joint i

$\mathcal{S}_j$ ..... set of joints for increased damping near singularity j

Variables

$\alpha_{s_j}, \alpha_{\ell_i}$ ..... damping factor as calculated for singularity j and for joint limit i

$w_{qs_{ij}}, w_{q\ell_i}$ ..... joint weight of joint i as calculated for singularity j and for joint limit i

d_{s_j}, d_{ℓ_i} ..... "distance" from singularity j and from joint limit i

u_{ℓ_i} ..... temporal ramp for joint limit i

δq_{iold} ..... differential for joint i from previous iteration

$$\alpha_{s_j} = \begin{cases} \alpha_0 (1-(d_{s_j}/d_{0s_j})^2) & \text{if } d_{s_j} < d_{0s_j} \\ 0 & \text{if } d_{s_j} \geq d_{0s_j} \end{cases} \tag{22}$$

$$w_{qs_{ij}} = \begin{cases} W(d_{s_j}) & \text{if } d_{s_j} < d_{0s_j} \text{ and } i \in \mathcal{S}_j \\ 1 & \text{if } d_{s_j} \geq d_{0s_j} \text{ or } i \notin \mathcal{S}_j \end{cases} \tag{23}$$

$$\text{where } W(d_{s_j}) = w_{q0s} + (1-w_{q0s}) d_{s_j}/d_{0s_j}$$

$$d_{\ell_i} = \begin{cases} hilim_i - q_i & \text{if } \delta q_{iold} \geq 0 \\ q_i - lolim_i & \text{if } \delta q_{iold} < 0 \end{cases} \tag{24}$$

$$u_{\ell_i} = \begin{cases} \max\{u_{\ell_i} - \delta u,\ d_{\ell_i}/d_{0\ell}\} & \text{if } d_{\ell_i} < d_{0\ell} \\ \min\{u_{\ell_i} + \delta u,\ 1\} & \text{if } d_{\ell_i} \geq d_{0\ell} \end{cases} \tag{25}$$

$$w_{q\ell_i} = w_{q0\ell} + (1-w_{q0\ell}) u_{\ell_i} \tag{26}$$

$$\alpha_{\ell_i} = \alpha_0 (1-u_{\ell_i}^2) \tag{27}$$

$$\alpha = \max\left\{ \max_j(\alpha_{s_j}),\ \max_i(\alpha_{\ell_i}) \right\} \tag{28}$$

$$w_{q_i} = \min\left\{ \min_j(w_{qs_{ij}}),\ w_{q\ell_i} \right\} \tag{29}$$

A constant $\mathbf{W_x}$ is used to normalize the Jacobian matrix. The first three rows of the Jacobian are related to position, which is measured in linear units, and the last three are related to orientation, which is measured in radians. The elements of $\mathbf{W_x}$ are set as follows:

$$\begin{aligned} w_{x_1} = w_{x_2} = w_{x_3} &= \text{NORM} \\ w_{x_4} = w_{x_5} = w_{x_6} &= 1 \end{aligned} \tag{30}$$

Here NORM $\equiv \pi/(\text{max reach of manipulator})$.

7 Discussion of an Implementation of Weighted Damped Least Squares

Weighted damped least squares, along with the scheme for setting the weights and damping factor, is used in the control of a slave manipulator in a

teleoperational master and slave system. The controller for the slave, which runs on a 33 MHz 486 PC/AT at about 300 Hz, includes: analog and digital interfaces, joint servos, inverse kinematics, and communications with the master controller. The telerobotic manipulators used in the system are the Kraft master and slave. These manipulators have six degrees of freedom and are kinematically similar. In the particular control system described however, they are not controlled using a joint space mapping between the master and slave, which is normally true of kinematically similar master and slave systems. Instead, a Cartesian space mapping, with scaling and indexing, is used. Therefore, the master can command unreachable configurations to the slave. Furthermore, the master may not be near a singularity, and therefore not hindered in any direction, while the slave is operating near a singularity, and therefore with limited capabilities in certain directions. Weighted damped least squares is used in dealing with these unreachable configurations and singularities, in real time within the local control loop of the slave. Exact solutions of equation (2) are found when the slave is away from joint limits and singularities, and approximate solutions are found when the slave is at or near a joint limit or singularity.

This inverse kinematics algorithm performs quite well in the system described above. The human operator is free to move the master about without worrying about what will happen when the slave is given physically unrealizable commands. The approximate solutions are both smooth and stable when the manipulator contacts the environment and/or when the manipulator is in several joint limit and/or singularity regions. Operation near a singularity results in approximate solutions with damped motion for the joints which swing about dangerously if the exact solution is used. Operation near joint limits result in approximate solutions that do not use the limited joint. The movements of the end effector, resulting from the approximate solutions, are intuitive to the operator. This is because the solutions are approximate in the task space. In contrast, solutions that are approximate in the joint space are not intuitive to the operator. Approximate joint space solutions result from an exact mathematical solution to equation (2) with partial implementation in joint space due to the physical limitations.

If the parameters of the algorithm are tuned well, the transitions between exact and approximate solutions and between different approximate solutions are smooth. However, if small values are used for d_{0s_j} and/or $d_{0\ell}$ then the manipulator tends to "jerk" when transitioning from one solution to the next. If a large value is used for $w_{q0\ell}$, then the resulting mathematical solution uses the limited joint even at the limit, and the physical solution, which is a partial implementation of the mathematical solution, is not intuitive. However, if zero is used for $w_{q0\ell}$ then the joint differential does not reverse and allow the joint move away from the limit. It was found that a small value of $w_{q0\ell}$ is sufficient to allow this to happen. It was also found experimentally that a small value for α_0 was sufficient. Although it is not done here, a minimum value for α_0 might be developed using some theoretical justification such as an upper bound on the condition number of $(\mathbf{J}\mathbf{J}^T + \alpha\mathbf{I})$ at the singularities.

8 Conclusion

In this paper a general procedure for the calculation of inverse kinematics with singularity and joint limit robustness has been presented. The procedure uses a damped least squares solution to solve the inverse kinematics iteratively and incorporates dynamic joint weighting for the reduction of specific joint differentials. The procedure gives an approximate solution at or near singularities and/or joint limits and an exact solution away from singularities and joint limits. An algorithm was given for the efficient calculation of the inverse kinematics using the procedure and a scheme for setting damping factor and joint weights was given. The algorithm and scheme were implemented for a six d.o.f. teleoperator and a well behaved slave manipulator resulted under teleoperational control.

References

1. Mayorga, R. V., and Milano, N., Wong, A. K. C., "A Fast Procedure for Manipulator Inverse Kinematics Computation and Singularities Prevention", *Journal of Robotic Systems*, Vol 10(1), pp 45-72, 1993.

2. Nakamura, Y., and Hanafusa, H., "Inverse Kinematic Solutions With Singularity Robustness for Robot Manipulator Control," *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol 108, pp 163-171, 1986.

3. Kelmar, L., and Kholsa, P. K., "Automatic Generation of Forward and Inverse Kinematics for a Reconfigurable Modular Manipulator System," *Journal of Robotic Systems*, Vol 7, pp 599-619, 1990.

4. Chan, S. K., and Lawrence, P. D., "General Inverse Kinematics with the Error Damped Pseudoinverse," *Proceedings of IEEE International Conference on Robotics and Automation*, pp 834-839, 1988.

5. Wampler, C. W., "Manipulator Inverse Kinematic Solutions Based on Vector Formulations and Damped Least-Squares Methods," *IEEE Transactions on Systems, Man, and Cybernetics*, vol SMC-16, No 1, pp 93-101, 1986.

6. Klein, C.A., and Huang, C. S., "Reveiw of Pseudoinverse Control for Use with Kinematically Redundant Manipulators," *IEEE Transactions on Systems, Man, and Cybernetics*, vol 13, pp 245-250, 1983.

7. Nakamura, Y., and Hanafusa, H., "Task Priority Based Redundancy Control of Robot Manipulators," *Second International Symposium on Robotics Research*, MIT Press, Cambridge MA, pp 155-162, 1985.

8. Walker, I. D., "The use of Kinematic Redundancy in Reducing Impact and Contact Effects in Manipulation," *Proceedings of IEEE International Conference on Robotics and Automation*, pp 434-439, 1990.

9. Hollerbach, J. M. and Suh, K. C., "Redundancy Resolution of Manipulators Through Torque Optimization," *Proceedings of IEEE International Conference on Robotics and Automation*, pp 1016-1021, 1985.

10. Craig, J. J., *Introduction to Robotics 2nd Edition*, Addison-Wesley, Reading MA, 1989.

11. Paul, R. P., *Robot Manipulators: Mathematics, Programming and Control*, The MIT Press, Cambridge MA, 1981.

12. Koivo, A. J., *Fundamentals for Control of Robotic Manipulators*, John Wiley & Sons, New York NY, 1989.

13. Pieper, D. L., "The Kinermatics of Manipulators under Computer Control," Ph.D. dissertation, Stanford University, 1969.

14. Whitney, D.E., "Resolved Motion Rate Control of Manipulators and Human Prostheses," *IEEE Transactions on Man-Machine Systems*, vol MMC-10, pp 47-53, 1969.

15. Stewart, G. W., *Introduction to Matrix Computations*, Academic Press, New York NY, 1973.

16. Dubey, R., and Luh, J. Y. S., "Redundant Robot Control for Higher Flexibility," *Proceedings of IEEE International Conference on Robotics and Automation*, pp 1066-1072, 1987.

17. Klein, C. A., "Use of Redundancy in the Design of Robotic Systems," *Second International Symposium on Robotics Research*, MIT Press, Cambridge MA, pp 207-214, 1985.

18. Ligeois, "Automatic Supervisory Control of the Configuration and Behaviour of Multibody Mechanisms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol 7, pp 868-871, 1977.

19. Yoshikawa, T., "Analysis and Control of Robot Manipulators with Redundancy," *First International Symposium on Robotics Research*, MIT Press, Cambridge MA, pp 735-747, 1984.

20. Yoshikawa, T., "Manipulability and Redundancy Control of Robotic Mechanisms," *Proceedings of IEEE International Conference on Robotics and Automation*, pp 1004-1009, 1985.

ControlShell: A Real-Time Software Framework

Stanley A. Schneider Vincent W. Chen Gerardo Pardo-Castellote

Real-Time Innovations, Inc.
954 Aster
Sunnyvale, California 94086

Aerospace Robotics Laboratory
Stanford University
Stanford, California 94305

Abstract

This paper describes ControlShell, *a next-generation CASE "framework" for real-time system software development. ControlShell's well-defined structure, graphical tools, and data management provide a unique component-based approach to real-time software generation and management. ControlShell is designed specifically to enable modular design and implementation of real-time software. By defining a set of interface specifications for inter-module interaction, ControlShell provides a basis for real-time code development and exchange.*

ControlShell includes many system-building tools, including a graphical data flow editor, a component data requirement editor, and a state-machine editor. It also includes a distributed data flow package, an execution configuration manager, a matrix package, and an object database and dynamic binding facility. ControlShell is being used in several applications, including the control of free-flying robots, underwater autonomous vehicles, and cooperating-arm robotic systems.

This paper presents an overview of the ControlShell architecture, and details the functions of several of the tools.

1 Introduction

Motivation System programs for real-time command and control are, for the most part, custom software. Emerging operating systems [1, 2, 3, 4, 5] provide some basic building blocks—scheduling, communication, etc.—but do not encourage or enable any structure on the application software. Information binding and flow control, event responses, sampled-data interfaces, network connectivity, user interfaces, etc. are all left to the programmer. As a result, each real-time system rapidly becomes a custom software implementation. With so many unique interfaces, even simple modules cannot be shared or reused.

An effective real-time framework must create a programming environment that facilitates sharing and reuse of real-time program modules. At a minimum, this requires providing interface specifications and data transfer mechanisms. The framework must also provide services and tools to combine modules and build systems from reusable components. Finally, the framework must meet the many challenges unique to real-time computing. For example:

- Real-time code must be able to react to external temporal events.
- The real-time execution environment is *fundamentally* multi-threaded and asynchronous.
- Real-time systems are usually composed of several different layers of control, each with different characteristics. For instance, strategic-level command and low-level servo control must be blended into a smoothly-operating system.
- Real-time systems must handle changing conditions, often requiring switching between drastically different modes of operation.
- Real-time systems are often physically distributed. In the simplest case, an operator control station may be remotely situated. More complex systems are comprised of many interacting distributed real-time and non-real-time subsystems.

All these challenges must be efficiently and smoothly handled by the architecture.

1.1 ControlShell's Solutions

Component-Based Design ControlShell is specifically designed to address these issues. ControlShell provides interface definitions and mechanisms for building real-time code modules. ControlShell also provides basic data structure specifications, and mechanisms for binding data with routines and specifying data-flow requirements. These two critical features make simple generic

Copyright c 1993 by Real-Time Innovations, Inc. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

packages (known as *components*) possible. ControlShell systems are built from combinations of these components.

An extensive library of pre-defined components is provided with the system, ranging from simple filters and controllers to complex trajectory generators and motion planning modules. New or custom components are easily added to the system via the graphical *Component Editor* (CE). The Component Editor allows simple specification of data interchange requirements. Code is automatically generated to permit instancing the new component into the system.

Graphical CASE System-Building Tools ControlShell also provides a set of powerful development tools for building complex systems. Building a system is accomplished by connecting components within a graphical *Data Flow Editor* (DFE). The data flow editor resolves the system data dependencies and orders the component modules for most efficient execution. Radical mode changes are supported via a "configuration manager" that permits quick reconfiguration of large numbers of active component routines.

Real-time systems also require higher-level control functions. ControlShell's event-driven finite state machine (FSM) capability provides easy strategic control. The state machine model features rule-based transition conditions, true callable sub-chain hierarchies, task synchronization and event management. A graphical FSM editor facilitates building state programs.

Real-Time System Services To provide support for real-time distributed systems, ControlShell includes a network connectivity package known as the *Network Data Delivery Service* (NDDS). NDDS provides distributed data flow. It naturally supports multiple anonymous data consumers and producers, arbitrary data types, and on-line reconfiguration and error recovery.

ControlShell also offers a database facility, direct support for sampled-data systems, a full matrix package, and an interactive menu system. Figure 1 presents an overview of the ControlShell toolset and design approach.

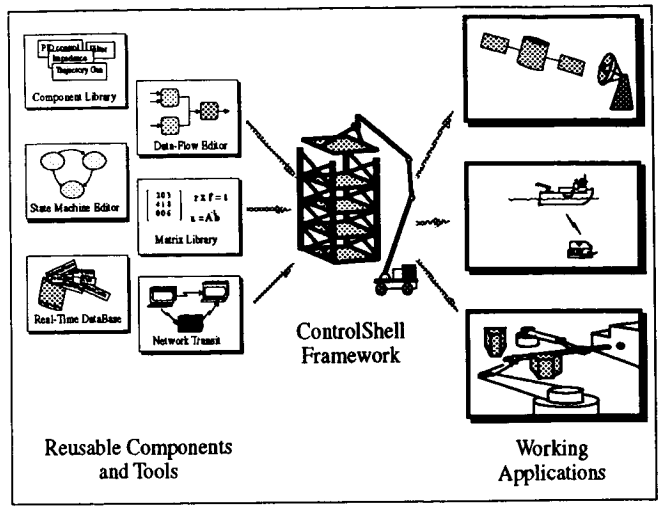


Figure 1: **ControlShell Design Process**

The ControlShell system designer uses the many powerful tools, system services, and prebuilt library components to construct a modular system.

2 Relation to Other Research

There are two quite different issues in real-time software system design:

- Hierarchy (what is communicated)
- Superstructure architecture (how it is communicated)

Several efforts are underway to define hierarchy specifications; NASREM is a notable example [6]. ControlShell makes no attempt to define hierarchical interfaces, but rather strives to provide a sufficiently generic software platform to allow the exploration of these issues. As such, this work takes a first step—defining the architecture superstructure (control and data flow mechanisms).

Several distributed data-flow architectures have been developed, including CMU's TCA/X [7, 8], Rice University's TelRIP [9], and Sparta's ARTSE [10]. These provide various levels of network services, but do not address repetitive service issues or resolve multiple data-producer conflicts in a symmetric robust "stateless" architecture as does the ControlShell NDDS system (see [11] for details). Also, they are not integrated within a general programming system.

Recently, more sophisticated programming environments have begun to emerge. For example, ORCAD [12] and COTS [13] are specialized robotics programming environments. Two commercial products, System Build with AutoCode from Integrated Systems, Inc. [14], and SIMULINK with C-Code Generation from the MathWorks, Inc. [15] are sophisticated control development environments. They offer easy-to-use rapid control system prototyping. They are not, however, architectures well suited to developing complex multi-layer distributed control hierarchies.

Implementation Experience ControlShell evolved from many years of research with real-time control systems. It was first developed for use with a multiple-arm cooperative robot project at Stanford University's Aerospace Robotics Laboratory[16, 17, 18]. From this start, ControlShell spread to become the basis for more

than 20 research projects in advanced control systems at Stanford. Among these were projects to study the control of flexible structures, adaptive control, control of mobile robots (including multiple coordinated robots), and high-bandwidth force control [19, 20, 21, 22, 23, 24]. More recently, a few industrial sites and two NASA centers have begun experimenting with ControlShell applications [25, 26]. ControlShell is now being jointly developed by Stanford University and Real-Time Innovations, Inc. It is supported under ARPA's Domain-Specific Software Architectures (DSSA) program.

This *continuous migration* from specific, working applications to wider spectrums of use is the key to usable generality. These applications continue to drive ControlShell's growth. To our knowledge, ControlShell is the only integrated framework package combining transparent networking, component-based system description, a state machine model, and a run-time executive.

3 Run-Time Structure

Some of the major system modules are shown in Figure 2. As shown in the figure, ControlShell is an *open* system, with application-accessible interfaces at each level. The figure is organized (loosely) into data and execution hierarchies.

At the lowest layer, ControlShell executes within the VxWorks real-time operating system environment. The simple base class known as *CSModules* is the building block for most executable constructs. Organizations of these modules, into lists, menus, and finite state machines form the core executable constructs. Users build useful execution-level atomic objects called *components* by defining derived classes from CSModules and binding them through the on-line data base to data matrices from the *CSMat* package. High-level graphical editors speed component definition, data flow specification and state machine programming. Network connectivity is provided by NDDS for all application modules.

4 Data Flow Design

Many real-time systems contain sampled-data subsystems. Here, we define a "sampled-data" system as any system with a clearly periodic nature. Common examples (each of which have been implemented under ControlShell) are digital control systems, real-time video image processing systems, and data acquisition systems. Each of these is characterized by a regular clock source.

Providing an environment where sampled-data program components can be interchanged is challenging. These programs have routines that must be executed during the sampling process, routines to initialize data structures (or hardware) when sampling begins, and perhaps to clean up when sampling ends. Further, many

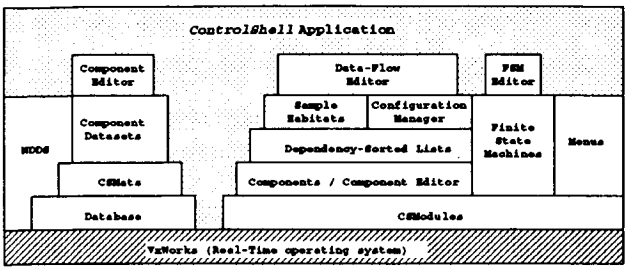


Figure 2: **Run-Time Structure**

ControlShell's open architecture provides many powerful services, while allowing application code free access to the underlying structures.

routines are dependent on knowledge of the timing parameters, etc. Although they may interact—say by passing data—sampled-data program components are often relatively independent. Requiring the application code to call each module's various routines directly destroys modularity.

4.1 Components

The *component* is the fundamental unit of reusable data-flow code in ControlShell. Components consist of one or more *sample modules* derived from CSModules. Sample modules have several pre-defined entry routines, including:

Routine	When executed
execute	Once each sample period
stateUpdate	After all executes are done
enable	When this module is made active
disable	When it is removed from the active list
startup	When sampling begins
shutdown	When sampling ends
timingChanged	When the sample rate changes
reset	When the user types "reset", or calls CSSampleReset
terminate	When the module is unloaded

Thus, a motor driver component might define a startup routine to initialize the hardware, an execute routine to control the motor, and a shutdown routine to disable the motors if sampling is interrupted for any reason. In addition, if any of its parameters depend on the sampling rate, it may request notification via a timingChanged method. By allowing components to attach easily to these critical times in the system, ControlShell defines an interface sufficient for installing (and therefore sharing) generic sampled-data programs.

Building Components: The Component Editor An easy-to-use graphical tool called the *component editor* (CE) assists the user in generating new components and specifying their data-flow interactions. The component editor defines all the input and output data requirements for the component, and creates a data type for the system to use when interacting with the component. The tool contains a code generator; it automatically generates a description of the component that the Data-Flow Editor can display (see below), and the code required to install instances of the component into ControlShell's run-time environment.

4.2 Execution Lists

An execution list is simply a dynamically changeable, ordered list of sample modules to be sequentially executed. The active set of modules on a list can be changed anytime. In fact, lists may drastically change their contents during system mode changes.

Execution lists may be sorted to provide automated run-time execution scheduling to resolve data dependencies. More specifically, the modules are sorted so that data consumers are always preceded by the appropriate data producers (see Figure 3). The system uses the specifications of the data flow requirements for each component to sort the dependencies and order the list. A side benefit of the sorting process is the error-checking that is performed to insure consistent data flow patterns.

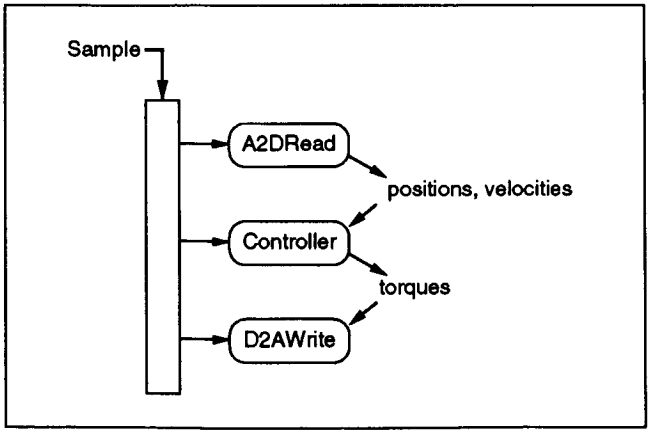


Figure 3: **Dependency-Sorted List**

Dependency-sorted execution lists provide automatic run-time sorting by data *dependencies.*

Sample Habitats ControlShell provides a named sampled-data environment, known as a *sample habitat.* A sample habitat encapsulates all the information and defines all the interfaces required for sampled-data programs to co-exist. It also contains routines to control the sampling process. For example, a module installed into a sample habitat can query its clock source and sample rate, start and stop the sampling process, etc.

Each sample habitat contains an independent task that executes the sample code. The task is clocked by the periodic source (such as a timer interrupt). Special components are provided to interface between habitats, allowing multi-rate controller designs.

4.3 Building Systems: The DFE Editor

Building systems of components is made simple by the graphical Data-Flow Editor (DFE). The DFE reads description files produced by the component editor, and then allows the user to connect components in a friendly graphical environment. It allows specification of all the data connections in the system, as well as reference inputs—gains, configuration constants and other parameters to the individual components. An example session is depicted in Figure 4.

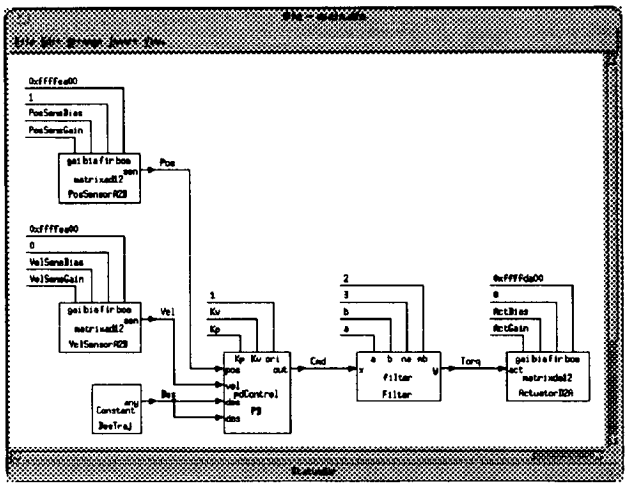


Figure 4: **Data-Flow Editor**

The data-flow editor builds collections of components into an executing system.

5 Configuration Management

Complex real-time systems often have to operate under many different conditions. The changing sets of conditions may require drastic changes in execution patterns. For example, a robotic system coming into contact with a hard surface may have to switch in a force control algorithm, along with its attendant sensor set, estimators, trajectory control routines, etc.

ControlShell's configuration manager directly supports this type of radical behavior change; it allows entire groups of modules to be quickly exchanged. Thus,

different system personalities can be easily interchanged during execution. This is a great boon during development, when an application programmer may wish, for example, to quickly compare controllers (See Figure 5). It is also of great utility in producing a multi-mode system design. By activating these changes from the state-machine facility (see below), the system is able to handle easily external events that cause major changes in system behavior.

Configuration Hierarchy The configuration manager essentially creates a four-level hierarchy of module groupings. Individual sample modules form the lowest level. These usually implement a single well-defined function. Sets of modules, called *module groups*, combine the simple functions implemented by single modules into complete executable subsystems.

Each module group is assigned to a *category*. One group in each installed category is said to be *active*, meaning its modules will be executed. Finally, a *configuration* is simply a specification of which group is active in each category.

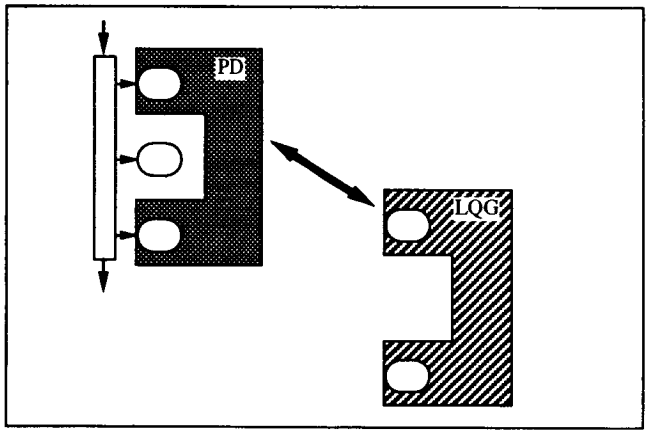


Figure 5: **Configuration Manager**

Configurations can be swapped in or out under program or menu control. This provides flexible run-time reconfiguration of the execution structure.

Example As a simple example, consider a system with two controllers: a proportional-plus-derivative controller named "PD", and an optimal controller known as "LQG". Suppose the PD controller requires filtered inputs, and thus consists of two sample modules: an instance of the *PDControl* component and a filter component. These two components would comprise the "PD" module group. The "LQG" controller module group may also be made up of several components. Both of these groups would be assigned to the category "controllers".

The user (or application code) can then easily switch controllers by changing the active module group in the "controller" category.

Now suppose further that the controllers require a more sophisticated sensor set. A category named "sensors" may also be defined, perhaps with module groups named "endpoint" and "joint". The highest level of the hierarchy allows the user to select an active group from each category, and name these selections as a *configuration*. Thus, the "JointPD" configuration might consist of the "joint" sensors and the "PD" controller. The "endptLQG" configuration could be the "endpoint" sensors and the "LQG" controller.

Category and Group Specification This subdivision may seem complex in these simple cases. However, it is quite powerful in more realistic systems. It has been shown to be quite natural in applications ranging from a vision-guided dual-arm robotic system able to catch moving objects [16] to flexible-beam adaptive controllers [27].

Assigning modules to groups and groups to categories is made quite simple with the ControlShell graphical DFE editor's "configuration definition" window, shown in Figure 6. New categories are added with the click of a button. To create a module group, the user simply names a group, and then clicks on the modules in the data-flow diagram that should belong to that group. The blocks are color-coded to relate the selections back to the user.

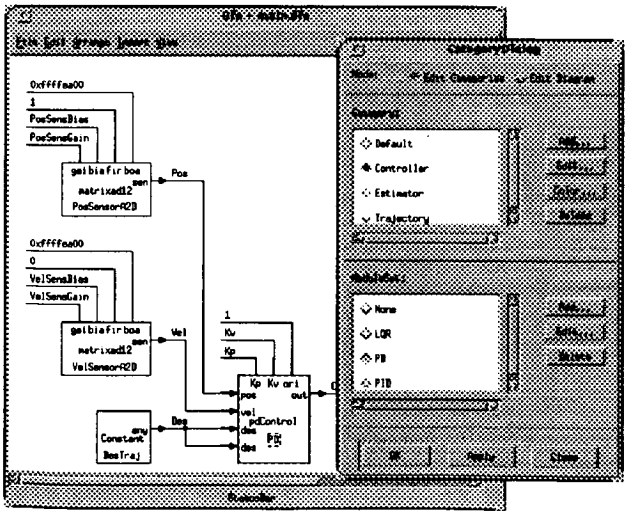


Figure 6: **Configuration Definition**

Configurations are easily defined within the DFE graphical interface.

6 Finite State Machines

A real-time system in the real world must operate in a complex, event-driven environment. With only a sequential programming language, the burden of managing and reacting to events is left to the programmer.

The Finite State Machine (FSM) module is designed to provide a simple strategic-level programming structure that also assists in managing events and concurrency in the system. The FSM module combines a non-sequential programming environment with natural event-driven process management. With this structure, the programmer is actively encouraged to divide the problem into small, independently executing processes.

To utilize the FSM module, the programmer first describes the task as a state transition graph. The graph can be directly described within ControlShell's graphical FSM editor (see Figure 7). Each transition—represented by an arrow in the graph—specifies a starting state, a boolean relation between stimuli that causes the transition, the CSModule to be executed when the transition occurs, and a series of "return code-next state" pairs that determine the program flow.

The FSM model is quite general; it supports rule-based transition conditions (reducing the number of states in complex systems), true callable sub-chains of states (so libraries of state subroutines can be developed), wild-card matching (so unexpected stimuli can be processed), global matching (allowing easy error processing), and conditional succession (so state programs may easily branch). Transitions are specified as boolean relations of three types of stimuli: transient, latched, and conditional. Transient stimuli have no value, and exist only instantaneously. Latched stimuli also have no value, but persist until some transition expression matches. Condition stimuli have string values; they persist indefinitely and thus represent memory in the system. Thus, the transition condition "Object = Visible AND Acquire" might cause a system to react to an acquisition command from a high-level controller. Providing these three stimuli types allows combination of both "system status" and "event" types of asynchronous inputs into easily-understood programs.

The FSM module takes advantage of the atomic message-passing capability of modern real-time kernels to weave the incoming asynchronous events into a single event stream. Any process can call a simple routine to queue the event; the FSM code spawns a process to execute the resulting event stream. The result is an easy-to-use, yet powerful real-time programming paradigm.

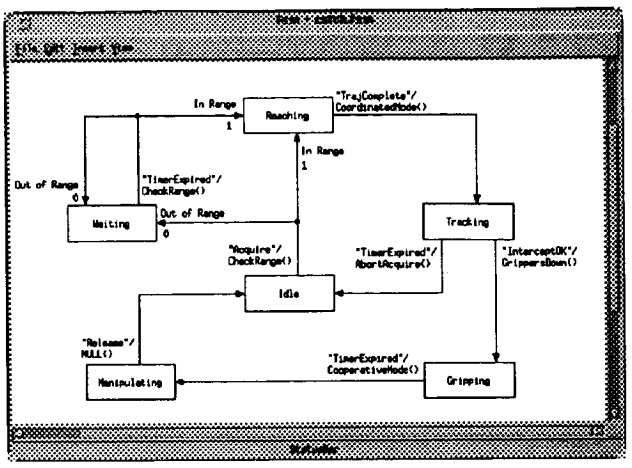


Figure 7: **Finite State Machine Editor**

State transition graphs allow easy visualization of multi-step operations; this example is a (simplified) program to catch a moving object with a dual-arm manipulator.

7 Data Control and Binding

Most data in a ControlShell application is embodied in *CSMats*. A CSMat is a named matrix of floating-point values. Each row and column of the matrix optionally contains a field name and a units specification. A complete real-time matrix mathematics utilities package is included. Components may combine multiple CSMats into structures for efficient reference and parameter passing.

The entire control hierarchies are created and bound to the correct data objects at run-time. The system is built from the graphically-generated description files produced by the DFE and FSM editors. This dynamic binding paradigm is very powerful—it combines the convenience of automatic system building with the flexibility of a dynamically changeable system. Thus, it provides the features of a full code generation without the pre-compiled inflexibility.

To support this dynamic binding, ControlShell incorporates a "linking" database facility. All instances of each data object (such as CSMats)—and each control construct (such as execution lists)—are entered into the database upon creation. The database allows "reference before creation" semantics for many object types; if a requested object is *not* in the database (i.e. it does not exist), an incomplete (e.g. zero-sized) object will be created by the database itself. This capability allows considerable flexibility at run-time; modules may, for instance, specify dependencies on data sets that do not yet exist, etc. Verification routines insure that the system is consistent before actual "live" execution begins.

8 Network Connectivity

ControlShell is integrated with a network connectivity package called the Network Data Delivery Service

(NDDS) [11]. NDDS is a novel network-transparent data-sharing system. NDDS features the ability to handle multiple producers, consumer update guarantees, notifications or "query" updates, dynamic binding of producers and consumers, user-defined data types, and more.

The NDDS system builds on the model of information producers (sources) and consumers (sinks). Producers register a set of data instances that they will produce and then "produce" the data at their own discretion. Consumers "subscribe" to updates of any data instances they require. Producers are unaware of prospective consumers; consumers are not concerned with who is producing the data they use. Thus, the network configuration can be easily changed as required. NDDS is a symmetric system, with no "special" or "privileged" nodes or name servers. All nodes are functionally identical and maintain their own databases. The routing protocol is connectionless and "quasi-stateless[1]"; all data producer and consumer information is dynamically maintained. Thus dropped packets, node failures, reconfigurations, over-rides, etc. are all handled naturally.

This scheme is particularly effective for systems (such as distributed control systems) where information is of a repetitive nature. NDDS is an efficient, easy-to-use distributed data-sharing system. Figure 8 illustrates the use of NDDS within a cooperating-arms robot system (see [24]).

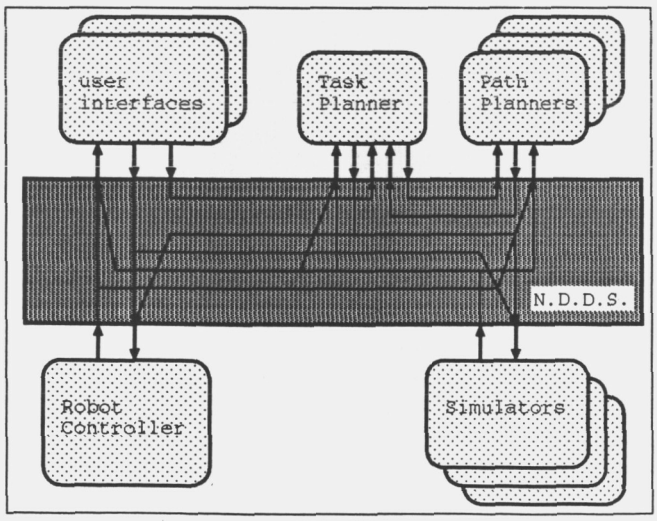


Figure 8: **Network Data Delivery Service**

NDDS provides a network "backplane". Each module can easily share data with any other module. The individual connections are handled transparently by the system.

[1] The databases at each node cache some state for efficiency, but all information decays over time.

9 Conclusions

This paper has presented a brief overview of the capabilities of the ControlShell system. ControlShell is designed—first and foremost—to be an environment that enables the development of complex real-time systems. Emphasis, therefore, has been placed on a clean and open system structure, powerful system-building tools, and inter-project code sharing and reuse.

Acknowledgements

ControlShell is being jointly developed by Stanford University and Real-Time Innovations, Inc. It is currently supported under ARPA's Domain-Specific Software Architectures (DSSA) program. The authors wish to expressly thank Dr. Marc Ullman for his many contributions to this project, and Dr. R. H. Cannon, Jr. for his guidance and leadership. The authors would also like to thank the many developers at Stanford, Loral, and NASA who have contributed ControlShell components.

References

[1] S. Narasimhan, D. Siegel, and J. M. Hollerbach, "Condor: A revised architecture for controlling the Utah/MIT hand," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Philadelphia, PA), pp. 446–449, April 1988.

[2] D. B. Stewart, D. E. Schmitz, and P. Khosla, "Chimera ii: A real-time multiprocessing environment for sensor-based robot control," in *Proceedings of the IEEE International Symposium on Intelligent Control*, (Albany, NY), September 1989.

[3] Wind River Systems, Inc., 1351 Ocean Ave., Emeryville, CA 94608, *VxWorks User's Manual*, 1988-1993.

[4] Software Components Group, Inc., 4655 Old Ironsides Drive, Santa Clara, CA 95054, *pSOS+/68K Real-Time, Multi-processing Operating System Kernel User's Manual*, 0.4.a ed., January 1989.

[5] Ready Systems, Inc., *VRTX User's Manual*, 1993.

[6] J. S. Albus, R. Lumia, and H. McCain, "Hierarchical control of intelligent machines applied to space station telerobots," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 24, pp. 535–541, September 1988.

[7] R. Simmons and C. Fedor, "Task control architecture programmer's guide," school of computer science / robotics istitute report, Carnegie Mellon University, 1992.

[8] C. Fedor, "TCX task communications," school of computer science / robotics istitute report, Carnegie Mellon University, 1993.

[9] J. D. Wise and L. Ciscon, *TelRIP Distributed Applications Environment Operating Manual.* Rice University, Houston Texas, 1992. Technical Report 9103.

[10] Sparta, Inc., 7926 Jones Branch Drive, McLean, VA 22102, *ARTSE product literature.*

[11] G. Pardo-Castellote and S. A. Schneider, "The network data delivery service," in *Proceedings of the International Conference on Robotics and Automation*, (San Diego, CA), IEEE, IEEE Computer Society, May 1994. (submitted).

[12] D. Simon, B. Espiau, E. Castillo, and K. Kapellos, "Computer-aided design of a generic robot controller handling reactivity and real-time control issues," programme 4 - robotique, image et vision, Unite De Recherche - INRIA-SOPHIA ANTIPOLIS, Domaine de Voluceau Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, France, November 1992.

[13] G. Hirzinger and J. Dietrich, "A computer architecture for intelligent machines," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Nice, France), May 1992.

[14] Integrated Systems, Inc., 2500 Mission College Boulevard, Santa Clara, CA 95054, *ISI Product Literature*, 1990-93.

[15] The MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick MA 01760, *Product Literature*, 1990-93.

[16] S. Schneider, *Experiments in the Dynamic and Strategic Control of Cooperating Manipulators*. PhD thesis, Stanford University, Stanford, CA 94305, September 1989. Also published as SUDAAR 586.

[17] S. Schneider and R. H. Cannon, "Object impedance control for cooperative manipulation: Theory and experimental results," *IEEE Journal of Robotics and Automation*, vol. 8, June 1992. Paper number B90145.

[18] S. A. Schneider and R. H. Cannon, "Experimental object-level strategic control with cooperating manipulators," *The International Journal of Robotics Research*, vol. 12, pp. 338–350, August 1993.

[19] R. Koningstein, *Experiments in Cooperative-Arm Object Manipulation with a Two-Armed Free-Flying Robot*. PhD thesis, Stanford University, Department of Aeronautics and Astronautics, Stanford, CA 94305, October 1990. Also published as SUDAAR 597.

[20] C. M. Oakley, *Experiments in Modelling and End-Point Control of Two-Link Flexible Manipulators*. PhD thesis, Stanford University, Department of Mechanical Engineering, Stanford, CA 94305, April 1991.

[21] C. R. Uhlik, *Experiments in High-Performance Nonlinear and Adaptive Control of a Two-Link, Flexible-Drive-Train Manipulator*. PhD thesis, Stanford University, Department of Electrical Engineering, Stanford, CA 94305, May 1990. Also published as SUDAAR 592.

[22] M. A. Ullman, *Experiments in Autonomous Navigation and Control of Multi-Manipulator Free-Flying Space Robots*. PhD thesis, Stanford University, Stanford, CA 94305, March 1993. Also published as SUDAAR 630.

[23] V. W. Chen, *Experiments in Adaptive Control of Multiple Cooperating Manipulators on a Free-Flying Space Robot*. PhD thesis, Stanford University, Stanford, CA 94305, December 1992. Also published as SUDAAR 631.

[24] G. Pardo-Castellote, T.-Y. Li, Y. Koga, R. H. C. Jr., J.-C. Latombe, and S. Schneider, "Experimental integration of planning in a distributed control system," in *Preprints of the Third International Symposium on Experimantal Robotics*, (Kyoto Japan), October 1993.

[25] S. W. Tilley, C. M. Francis, K. Emerick, and M. G. Hollars, "Preliminary results on noncolocated torque control of space robot actuators," in *Proceedings of the NASA Conference on Space Telerobotics*, (Pasadena, CA), NASA, February 1989.

[26] S. W. Tilley, M. G. Hollars, and K. S. Emerick, "Experimental control results in a compact space robot actuator," in *Proceedings of the ASME Winter Annual Meeting*, (San Francisco, CA), December 1989.

[27] L. Alder, *Control of a Flexible-Link Robotic Arm Manipulating An Unknown Dynamic Payload*. PhD thesis, Stanford University, Stanford, CA 94305, February 1993. Also published as SUDAAR 632.

LYNDON B. JOHNSON SPACE CENTER (JSC) PROPOSED DUAL-USE TECHNOLOGY INVESTMENT PROGRAM IN INTELLIGENT ROBOTICS

Jon D. Erickson*
National Aeronautics and Space Administration Lyndon B. Johnson Space Center
Houston, Texas 77058

Abstract

This paper presents an overview of the proposed Lyndon B. Johnson Space Center (JSC) precompetitive, dual-use technology investment project in robotics. New robotic technology in advanced robots, which can recognize and respond to their environments and to spoken human supervision so as to perform a variety of combined mobility and manipulation tasks in various sectors, is an objective of this work. In the U.S. economy, such robots offer the benefits of improved global competitiveness in a critical industrial sector; improved productivity by the end users of these robots; a growing robotics industry that produces jobs and profits; lower cost health care delivery with quality improvements; and, as these "intelligent" robots become acceptable throughout society, an increase in the standard of living for everyone. In space, such robots will provide improved safety, reliability, and productivity as Space Station evolves, and will enable human space exploration (by human/robot teams).

The proposed effort consists of partnerships between manufacturers, universities, and JSC to develop working production prototypes of these robots by leveraging current development by both sides. Currently targeted applications are in the manufacturing, health care, services, and construction sectors of the U.S. economy and in the inspection, servicing, maintenance, and repair aspects of space exploration. But the focus is on the generic software architecture and standardized interfaces for custom modules tailored for the various applications allowing end users to customize a robot as PC users customize PC's. Production prototypes would be completed in 5 years under this proposal.

* Chief Scientist, Automation and Robotics Division, Member AIAA

Copyright © 1994 by the American Institute of Aeronautics and Astronautics, Inc. No copyright is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all right under the copyright claimed herein for Governmental purposes. All other rights are reserved by the copyright owner.

1. Introduction

This paper suggests a large number of opportunities for robotic manufacturers, integrators, potential buyers/users of robots, commercial technology developers, and universities to work with the NASA JSC Automation and Robotics Division, with NASA funding a major portion of the development. The focus is intelligent robotics as partial solutions to productivity problems in several sectors of application. The stage of development addressed is precommercial. In each case dual use is a prerequisite: there must be a space use as well as a nonspace, commercial use. Generally, this is easily the case.

The specific motivation and rationale for this NASA JSC proposed technology investment program is detailed in Erickson [1]. The general policy that sets the context for the NASA technology investment program, which will begin in 1994, is given in Clinton and Gore [2].

It is important to understand that although a set of objectives, an approach, and a number of tasks are suggested here, these are meant to stimulate the creative thought process of those in nonaerospace and aerospace industry to propose objectives, approaches, and tasks that they believe, due to their involvement with their commercial buyers/users, would be economic and profitable as a result of jointly funded developmental efforts with NASA.

Intelligent robotics is the use of robotic systems in solving problems in tasks and environments where the robot's ability to acquire and apply knowledge and skills to achieve stated goals in the face of variations, difficulties, and complexities imposed by a dynamic environment having significant unpredictability is crucial to success. This means the robots can recognize and respond to their environments at the pace of their environments and to spoken human supervision so as to

perform a variety of mobility and manipulation tasks. This does not require broad-based general intelligence or common sense by the robot.

These robots are capable of significant, autonomous reaction to unpredictable events, yet are subject to optional human supervision during operation in a natural way, such as by voice. We refer to this capability in the supervised robot as "adjustable autonomy." Also, a key essence is that previously acquired knowledge is combined with knowledge acquired at the instant of task performance.

The overall approach can be summarized as capitalizing on a software architecture that can be viewed as generic and modular, and hardware approaches that are modular, reconfigurable, and extendible. Many of the software modules, such as a deliberative planner, world model, and natural language interface, can also be viewed as generic. Other software is bundled with certain hardware; e.g., sensing software is bundled with specific sensor hardware. This leads to the concept of a modular, end-user customized robot, put together from modules with standard interfaces[3,4,5] such as users do with a personal computer. An integrated computer aided concurrent engineering environment that we are working on[6] is a way to achieve close teamwork by geographically distributed "virtual" teams to develop the production prototypes.

JSC can be a key partner in this dual-use technology investment program in intelligent robotics for two reasons: (1) human space exploration missions require supervised intelligent robotics as enabling tools[7,8] and, hence, must develop or have developed supervised intelligent robotic systems and (2) intelligent robotic technology is being developed for space applications at JSC (but has a strong crosscutting or generic flavor) that is advancing the state of the art[9,10] and is producing both skilled personnel and adaptable developmental infrastructure, such as low cost simulation environments for software testing and integrated testbeds for complete prototype testing. JSC also has a Small Business Innovative Research (SBIR) program[11] for intelligent robotics, which is underutilized and has no commercial cost sharing requirement. It is limited in scope to about $0.6 million and 2 years in Phase II efforts.

A key element in the cutting edge intelligent robotics technology work at JSC is an understanding of and solution approach to the key issue of melding artificial intelligence planners with reactive capabilities. Artificial intelligence planners offer goal-achieving planning, but also high-time variance due to searching. Reactive capabilities are needed to deal safely in real time with dynamic, unpredictable environments at the pace of the dynamics[9]. A second key element that JSC brings is an approach to improved robotic reliability as required for space, but also useful in industry. A third key element that JSC brings to cutting edge technology is an understanding of and solution approach to the key issue of robotic safety while maintaining productivity.

Of all these elements, the personnel skilled in the state of the art and knowledgeable about the technology are the most important.

2. Overview of Proposed Activities

New robotic technology in advanced robots that can recognize and respond to their environments and to spoken human supervision so as to perform a variety of mobility and manipulation tasks in various sectors is an objective of this proposed effort. In the U.S. economy, such robots offer the benefits of improved global competitiveness in a critical industrial sector; improved productivity by the end users of these robots; a growing robotics industry that produces jobs and profits; lower cost health care delivery with quality improvements; and, as these "intelligent" robots become acceptable throughout society, an increase in the standard of living for everyone[12]. In space, such robots will provide improved safety, reliability, and productivity as Space Station evolves, and will enable human space exploration (by human/robot teams).

The proposed effort consists of partnerships between manufacturers, users, universities, and JSC to develop working production prototypes of these robots by leveraging current development by manufacturers and JSC. Currently targeted applications are in the manufacturing, health care, services, and construction sectors of the U.S. economy and in the inspection, servicing, maintenance, and repair aspects of space exploration. But the focus is on the generic software architecture and standardized interfaces for custom modules tailored for the various applications, allowing end users to customize a robot as personal computer users customize PC's. Production prototypes would be completed in 5 years under this proposal, as would automated developmental environments and integrated testbeds.

JSC possesses the required core skills in its civil service and contractors to form the nucleus of the multiple partnerships. Current technology integration efforts at JSC include the EVA helper/retriever supervised intelligent robot [10], the mobile robotics testbed project, and the Soda-Pup entry in the AAAI national robotics competition (1992 award winner). In addition, JSC is responsible for engineering upgrades to the Shuttle Remote Manipulator Systems, integration of the Mobile Servicing Systems and Special Purpose Dexterous Manipulator into Space Station, and numerous robotics technology efforts.

User coordination involves interested manufacturers with deployed robots. Joint facility sharing and temporary personnel exchange are possible.

The overall set of activities has been grouped into the following seven related categories of tasks, each with its own objectives and approach.

1. Problem-Solving Insertion of Robot Intelligence Technology
2. Generic Intelligent Robotics Software Architecture
3. Modular Manipulation and Mobility for Robotics
4. Integrated Sensing and Perception Capabilities for Robotics
5. Robotic Surrogates for Human Grasping and Manipulation
6. Integrated Prototyping Environment for Robotics
7. Robotic Applications in Advanced Manufacturing, Health Care, Service Industries, and Construction

The following sections present the objectives, approach, and benefits for each of these categories of tasks and give the titles of the set of tasks grouped into that category. One-page task descriptions are available [13] for all tasks, giving task objectives, proposed effort, major milestones, benefits, and other information.

Problem-Solving Insertion of Robot Intelligence Technology

The objectives are (1) to work as a team with end user industries whose productivity problems can be solved by integrating adaptive robots into the advanced manufacturing or service process, and in so doing to develop a new paradigm of product line development for robot manufacturers and (2) to provide the robotics industry sensor/software control techniques that will make the robots more flexible and attractive for use by end user industries. This will impact the end users of these robots by improving the end users' efficiency and productivity and thus improved global competitiveness. This will also stimulate robot demand and provide a new way of doing business for robot manufacturers. The benefits for space will be a healthier robotics industry capable of supplying quality robotics at lower costs.

The proposed effort consists of partnerships between manufacturers, integrators, nonprofits, and JSC to solve end user problems by integrating adaptive robots into end user operations. As part of that effort the robot manufacturers' products must first be upgraded with sensing and intelligent reaction capabilities from new sensor/software control technology. A key product is the development, documentation, and refinement of the problem-solving insertion process for intelligent robotics technology, including end user problem identification techniques; problem selection criteria; requirements definition; development of a solution; integration with the end user people, processes, and equipment; user training; and continuing user support.

In related work, JSC is responsible for integration of the Mobile Servicing System and Special Purpose Dexterous Manipulator into Space Station, which gives us the necessary experience and insight to help users.

The eight tasks in this category are the following:

- End User Target Problems Identification
- End User Problem Selection
- Selected Problem Requirements Definition
- Design, Development, Test, and Evaluation of Solution
- Integration with User Equipment, Processes, and People
- User Training Definition
- Continuing User Support Definition
- Problem Solving Insertion Process Development, Documentation, and Refinement

The benefits of this problem-solving insertion are that the end user businesses obtain a useful

solution to their problems. The robot manufacturers and integrators obtain a better understanding of the integration process, not only as part of the problem-solving insertion of their products but also as part of the requirements for capabilities in their products. JSC gets a benefit for space applications due to understanding of capabilities of intelligent robots required to solve certain types of problems.

Generic Intelligent Robotics Software Architecture

The objective is a generic, supervised intelligent robotics software architecture that provides a portable software approach that integrates intelligent planning and reactive control with sensing and internal representation of environment to enable advanced robots that can recognize and respond to their surroundings and to spoken human supervision in order to perform a variety of manipulation and mobility tasks. The benefits of such an architecture are the faster development time, lower cost, and increased adaptable and flexible performance. In turn, these provide improved productivity by the end users of these robots, whether terrestrial or space.

The proposed effort consists of partnerships between manufacturers, nonprofits, and JSC to develop the software and evaluate its characteristics and robustness in several tasks and environments. The design of the software architecture, which is the framework (functional decomposition) that integrates the separate functional modules into a coherent system, is dictated in large measure by the tasks and nature of the environment. Because both the goal-achieving tasks and the partially unpredictable nature of the environments are similar on Earth and in space, the software architecture can be viewed as generic. Many of the software modules, such as a deliberative planner, world model capability, and natural language interface, can also be viewed as generic. Other software is bundled with certain hardware; e.g., sensing software is bundled with specific sensor hardware.

Current work on the EVA helper/retriever supervised intelligent robot, the mobile robotics testbed project, and Soda-Pup project at JSC have provided us the necessary insight and experience. Not just any architecture will do here. It must solve the key issue of combining deliberative goal-achieving planning with reactive capabilities in such a way as not to limit the intelligence of the planner or the safety of the reactive execution. The JSC work is believed to offer such a solution. It is a practical implementation of the mathematical theory of intelligent robots [14].

The ten tasks in this category are the following:

- Artificial Intelligence Planning Software
- Sequencing and Scheduling Software
- Reactive Controller Software
- Integration of Natural Language Understanding Into Architecture
- Real-Time Speech Planning Software
- World Modeling Software
- Software Development Environment
- Integrated Software Architecture
- Integrated Testing Against Simulated Environments
- Skill Acquisition

A generic software architecture for supervised intelligent robots will enable portability and reuse, major time and cost savings in development and testing, more robust and higher quality software, and maintenance and training cost reductions. People will have a natural means of supervision by including task limited natural language understanding and speech generation software in the robotics software architecture. Improved safety of operations is also a benefit. These benefits apply in space and in the U.S. economy.

Modular Manipulation and Mobility for Robotics

The objective here is to develop a set of standardized modular components that can be reconfigured as required into modular robots offering a broad spectrum of tasks, reduced system costs, reduced weight, reduced mean time to repair, changeout of broken components, and reduced operator training. As components for an integrated prototyping environment for evaluating alternate approaches to design of robotic systems, these contribute to making adaptive robots "faster, better, and cheaper."

The proposed effort consists of partnerships with robotic manufacturers, nonprofits, and universities to develop working production prototypes of a set of standardized modular components. The development of standards for mechanical and electrical connections and similar

modular interfaces will be a product as well. Both manipulator and mobility systems with robot body structures would be developed. Arm sockets, links, joints, actuators, and sensors would be designed and developed to standards for manipulators. Wheels, tracks, suspension, drive train motors, gears, brakes, drive control electronics, structure, pan/tilt units, power and communications subsystems, and sensors would be designed and developed to standards for mobility and body systems.

We have a current effort in designing modular components for space manipulation [4].

The eight tasks in this category are the following:

- Manipulator Socket, Link, Joint, Actuator, and Sensor Modular Component Standards Development
- Manipulator Modular Component Designs
- Manipulator Modular Component Development
- Mobility Modular Component Standards Development
- Mobility Modular Component Designs
- Mobility Modular Component Development
- Modular Robotics Testbed Development
- Modular Prototype Testing on the Testbed

Modular, reconfigurable manipulator arm and mobility subsystems as part of modular, reconfigurable intelligent robots will reduce cost, reduce development time and cost, enable more uses through reuse and reconfiguration, reduce maintenance and repair time and costs, and increase availability (uptime). This approach also enables low cost, rapid prototyping, and rapid development of intelligent robots with testing against intended tasks and environments to improve quality. All applications are beneficial, especially those for space.

Integrated Sensing and Perception Capabilities for Robotics

Development of the capability to select and tailor sensors and real-time perception processing to the task- and environment-driven requirements of adaptive robots is the objective of this portion of the effort. Perception is the extraction of useful information about the environment needed to understand the situation to complete the task successfully.

These capabilities must be integrated with the interface standards of a generic, supervised intelligent robotics software architecture. These are the most important capabilities enabling reactive behavior and deliberative, goal-achieving planning and actions. By enabling advanced robots to recognize their dynamic environments so as to respond appropriately, this effort leads to improved productivity by the end users of these robots, a growing robotics industry that produces jobs and profits, and improved global competitiveness. In space, these capabilities enable robots to provide the flexible support that enables space exploration (by human/robot teams).

Integration of sensing and perception into planning and control in a robust way is a challenge for at least two fundamental reasons. First, the time available to sense and perceive the many dynamic and unpredictable elements of the situation is limited. Second, perception attaches meaning to the link between a conception of the environment and the objective environment. Perception is the process of inference that recognizes regularities in sensor data that are known on the basis of a model of the world to be reliably related to causal structure of objects and their relations in the environment and then conveys this to cognition. Sensory data underdetermines world structure; therefore, a model of world structure is required.

Perception involves understanding generic, generally applicable models of world structure (not merely specific object models) and how that causal structure evidences itself in sensor data. Causal structure is of interest so as to be able to predict consequences, anticipate events, and plan actions so as to achieve goals. Perception is generally focussed by needs for information that supports planning and reasoning for goals achievement. Designing perception involves converting the understanding and inference processes into calculational steps (algorithms and inferences) and designing computation hardware systems to meet the requirements of information at rates and latencies required to deal with a dynamic environment.

The proposed effort consists of partnerships between manufacturers, nonprofits, universities, and JSC to develop a set of sensors and perception processing appropriate to numerous task- and environment-driven requirements for adaptive robot applications, both in the U.S. economy and in space. Included here are vision sensing and visual perception, along with speech recognition

and task limited natural language understanding (speech perception). The unification of visual and speech perception is also included here. Proximity sensing, tactile/slip sensing, and force/torque sensing, which are critical aspects of many manipulation tasks, are addressed in the next category of tasks.

Current sensing and perception efforts at JSC include focused developments for EVA helper/retriever (laser scanner, stereo video, torque and proximity sensors, speech recognition and task limited, natural language understanding, etc.) and the mobile robotics testbed project (real-time stereo vision).

The six tasks in this category are the following:

- Vision Sensors and Sensing Software Development
- Finding, Recognizing, Locating, and Tracking Objects and Humans
- Visual Perception of Objects' Spatial Relations
- Visual Perception of Objects' Condition and Process Participation
- Speech Recognition and Natural Language Understanding
- Unification of Visual and Speech Perception

The benefit of sensing and perception capabilities is to enable the supervised intelligent robot both to extract needed information about the changing task environment, including humans, on a real-time basis so as to react safely and appropriately, and to build and continuously update internal representations of the changing environments so as to plan safe goal-achieving actions. People will have a natural means of supervision through task limited, natural language understanding software. The unification of visual and speech perception adds power to the human/robot team. These benefits apply in space and in the U.S. terrestrial economy.

Robotic Surrogates for Human Grasping and Manipulation

Robots and humans must be capable of interacting with the same environment in terms of grasping and manipulation for certain tasks. Dexterous robotic grasping and manipulation capability must be developed to achieve this capability. The robot may operate in conjunction with a human as an apprentice or may be substituted for a human (e.g., in hazardous operations). The benefits to the U.S. economy from robots with such capability would be very large: improved global competitiveness; improved productivity by the end users; a growing robotics industry meaning more jobs and profits; and an increased standard of living in the United States. In space, robots with these capabilities are required to interface with space hardware on astronaut/robot teams. This would reduce the cost of designing the robotic environment and allow more tasks to be done robotically.

The proposed effort consists of partnerships between manufacturers and JSC to develop working production prototypes of human-scale versions of robot hands by leveraging current development by both sides. Integration of tactile, slip, force, and torque sensing; adaptive grasping; stable grasp recognition; and manipulation strategy approaches will be accomplished. However, it should be recognized that the resulting robot hands are not expected to be equivalent to human hands. Limited multitask capability is all that is expected in the 5-year term of this effort.

EVA helper/retriever and the dexterous, anthropomorphic robotic testbed (DART) are two of the current related efforts at JSC, as well as some SBIR developments.

The nine tasks in this category are the following:

- Hand designs
- Integrated hand, wrist, and arm designs
- Tactile/slip sensors, sensing software, and perception software
- Proximity sensors, sensing software, and perception software
- Force/torque sensors, sensing software, and perception software
- Integrated sensing with hand, wrist, arm to provide stable grasp recognition and other intelligent functions
- Grasping and manipulation strategies
- Collision avoidance strategies
- Compliance strategies

Supervised intelligent robots and human ability to interact with the same environment in terms of dexterous grasping and manipulation will provide major benefits in U.S. industry, service applications, and in space. Costly special designs and structuring of the robot environment will be

minimized or eliminated, thus reducing costs. Robots will be able to operate in conjunction with humans as robot apprentices to humans on human/robot teams.

An Integrated Prototyping Environment for Robotics

The objective is to develop an integrated rapid prototyping and rapid development environment for building robotic systems "faster, better, and cheaper" based on modularity, reconfigurability, and extendibility, including a library of hardware modules (such as manipulators, tools, and sensors), complementary software modules (such as sensing and perception strategies and manipulator control), and software advisors designed to reduce the cost of programming robots. This effort would also leverage development of a generic intelligent robotics software architecture in a related subcategory.

The proposed effort consists of partnerships between manufacturers and JSC to develop an integrated prototyping environment that will allow users to generate and evaluate alternate approaches to the design of a robotic system quickly. This effort would also leverage development of modular manipulation and mobility for robots and other related subcategories.

Task-directed process design with a systems engineering focus and reconfigurable modular designs are strong points of our experience at JSC that are critical to success here.

The five tasks in this category are the following:

- Requirements for Prototyping Environment
- Design of Prototyping Environment
- Development of Prototyping Environment
- Testing of Prototyping Environment
- Knowledge Support System

Automation of the process of designing and developing intelligent robots reduces costs and development time. Automation of the testing of intelligent robots also reduces costs and development time while providing the user early feedback that the robots will solve the problems. All markets benefit: advanced manufacturing, health care, service industries, construction, mining, space, etc.

Robotic Applications in Advanced Manufacturing, Health Care, Service Industries, and Construction

This effort's objectives are to enable the manufacture and marketing of supervised intelligent robotic systems for applications in advanced manufacturing, health care, service industries, and construction by developing working production prototypes. Production prototypes will also be developed for inspection, servicing, maintenance, and repair tasks for space exploration. Such advanced robotic systems offer the benefits of improved productivity by the end users and improved global competitiveness to the U.S. economy. In space, such robots provide improved safety, reliability, and productivity as Space Station evolves and enables human space exploration (by human/robot teams).

The proposed effort consists of partnerships between manufacturers, nonprofits, doctors and hospitals, universities, and JSC.

The required core skills are available at JSC in its civil service and contractors to form the nucleus of the multiple partnerships. Current technology integration efforts include the EVA helper/retriever supervised intelligent robot, the mobile robotics testbed project, and the Soda-Pup entry in the AAAI national robotics competition. In addition, JSC is responsible for numerous applications of robotics.

In our ongoing relationship with the Texas Medical Center, recent interest by Drs. Steve Kroll and Chuck Van Duren in robotic microsurgery and arterial catheterization has been shown.

The four tasks in this category are the following:

- Robotic Applications in Advanced Manufacturing
- Robotic Applications in Health Care
- Robotic Applications in Service Industries
- Robotic Applications in Construction

The benefits of intelligent robots to advanced manufacturing are spelled out in detail in Erickson [1]. The benefit to health care is lower cost health care delivery with quality improvements due to improvements in productivity. The benefits to other service industry applications are improvements in productivity. The benefits of intelligent robots to construction include improved construction time and productivity.

3. Concluding Remarks

We have presented a "straw man" pre-competitive, dual use technology program in intelligent robotics intended to stimulate the creative aspects of nonaerospace and aerospace industry to propose their own objectives, approaches, and tasks for new jointly funded partnerships with NASA JSC. It is evidence of our "earnest" and that we are ready to proceed with our end of the partnerships.

4. References

1. Erickson, J.D. "Intelligent Robotics Can Boost America's Economic Growth." Proceedings of AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space, Houston, TX, Mar. 1994.

2. Clinton, William J.; and Gore, Albert, Jr. "Technology for America's Economic Growth, A New Direction to Build Economic Strength." White House, Washington, DC, Feb. 22, 1993.

3. Bonasso, R.P.; and Slack, M.G. "Ideas on a System Design for End-User Robots." Proceedings of SPIE 1829, Cooperative Intelligent Robotics in Space III, Boston, MA, Nov. 1992.

4. Ambrose, R.O.; Aalund, M.P.; and Tesar, D. "Designing Modular Robots for a Spectrum of Space Operations." Proceedings of SPIE 1829, Cooperative Intelligent Robotics in Space III, Boston, MA, Nov.1992.

5. daCosta, F.; et al. "An Integrated Prototyping Environment for Programmable Automation." Proceedings of SPIE 1829, Cooperative Intelligent Robotics in Space III, Boston, MA, Nov. 1992.

6. Erickson, J.D.; and Lawler, D. "Integrated Computer Aided Concurrent Engineering." Dual-Use Space Technology Conference and Exhibition, NASA JSC, Feb. 1994.

7. Erickson, J.D. "Supervised Space Robots Are Needed in Space Exploration." Proceedings AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space, Houston, TX, Mar. 1994.

8. Miles, G.E.; and Erickson, J.D. "Robotics in Controlled Environment Agriculture." Proceedings of AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space, Houston, TX, Mar. 1994.

9. Erickson, J.D.; et al. "An Intelligent Space Robot for Crew Help and Crew and Equipment Retrieval." International Journal of Applied Intelligence, accepted for publication in 1994.

10. Erickson, J. D.; Grimm, K.A.; and Pendleton, T.W. "An Intelligent Robot for Helping Astronauts." Proceedings of AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space, Houston, TX, Mar. 1994.

11. NASA Small Business Innovation Research Program Solicitation, SBIR 93-1. NASA Headquarters, Washington, DC, May 1993.

12. Office of Technology Assessment. "Exploring the Moon and Mars: Choices for the Nation." Congress of the U.S., Washington, DC, Aug. 1991.

13. Erickson, J.D. "NASA JSC Proposed Dual Use Technology Investment Program in Robotics." JSC-37922, Automation and Robotics Division, NASA JSC, Nov. 1993.

14. Saridis, G.N. "Analytic Formulation of the Principle of Increasing Precision with Decreasing Intelligence for Intelligent Machines." Automatica, Vol.25, Nov. 3, 1989.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington , DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Mar/94	Conference Publication

4. TITLE AND SUBTITLE
AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service,and Space

5. FUNDING NUMBERS

6. AUTHOR(S)
Jon D. Erickson, Editor

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Lyndon B. Johnson Space Center
Houston, Texas 77058

8. PERFORMING ORGANIZATION REPORT NUMBERS
S-754

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
American Institute of Aeronautics & Astronautics
Washington, D.C. 20073
National Aeronautics and Space Admistration
Washington, D.C. 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
CP-3251

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
unclassified/unlimited
Available from the NASA Center for AeroSpace Information, 800 Elkridge Landing Road, Linthicum Heights, MD, 21090; (301) 621-0390.

Subject Category: 63

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS '94) was originally proposed because of the strong belief that America's problems of global economic competitiveness and job creation and preservation can partly be solved by the use of intelligent robotics, which are also required for human space exploration missions. It was also recognized that in the applications-driven approach there are a far greater set of common problems and solution approaches in field, factory, service, and space applications to be leveraged for time and cost savings than the differences in details would lead one to believe. This insight couple with a sense of national urgency made a continuing series of conferences to share the details of the common problems and solutions across these different fields not only a natural step, but a necessary one. Further, it was recognized that a strong focusing effort is needed to move from recent factory-based technology into robotic systems with sufficient intelligence, reliability, safety, flexibility, and human/machine interoperability to meet the rigorous demands of these fields, the scope of which is beyond the capability of any one area.

The papers in these proceedings are evidence that users in each field, manufacturers and integrators, and technology developers are rapidly increasing their understanding of integrating robotic systems on Earth and in space to accomplish economically important tasks requiring mobility and manipulation. The 21 sessions of technical papers in 7 tracks plus 2 plenary sessions cover just the tip of this major progress, but reveal its presence nonetheless.

14. SUBJECT TERMS
robotics, robots, computers, computer programming, artificial intelligence

15. NUMBER OF PAGES
923

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	UL